SESSION 6

iOS DEVELOPMENT

REQUIREMENTS FOR

FINAL PROJECTS

UNDERSTANDING

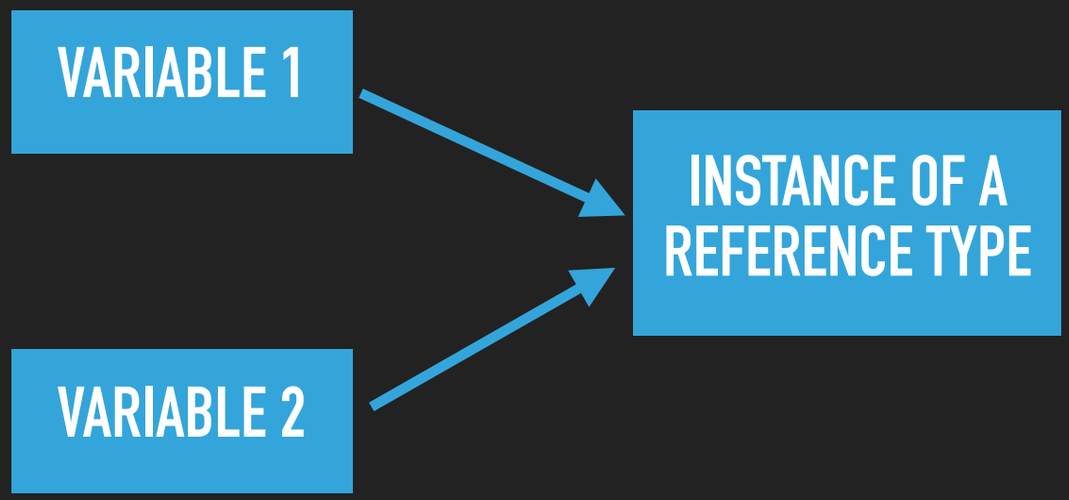
MEMORY MANAGEMENT

REFERENCE VS. VALUE

- Classes are reference types
- Structs and enums are value types

REFERENCE VS. VALUE

- Reference types: copying creates a shared instance



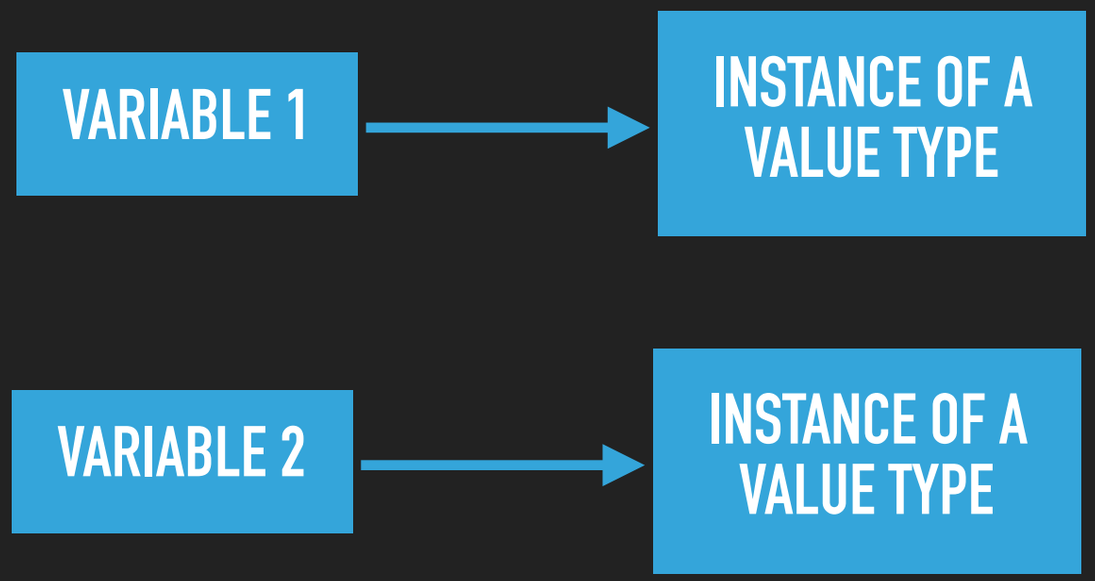
REFERENCE VS. VALUE

- When two variables point to the same instance, changing one changes the other

```
// Reference Types

class City {
    var isBelowFreezing = false
}

let chicago = City()
let windyCity = chicago

chicago.isBelowFreezing = true

chicago.isBelowFreezing // true
windyCity.isBelowFreezing // true

```

REFERENCE VS. VALUE

- Value types: copying creates a new instance with its own data



REFERENCE VS. VALUE

- Value types: changing one does not change the other

```
18  // Value Types
19
20  struct Forecast {
21      var degreesFarenheit: Int = 32
22  }
23
24  var forecastToday = Forecast()
25  var forecastTomorrow = forecastToday
26
27  forecastToday.degreesFarenheit += 5
28
29  forecastToday.degreesFarenheit // 37
30  forecastTomorrow.degreesFarenheit // 32
```

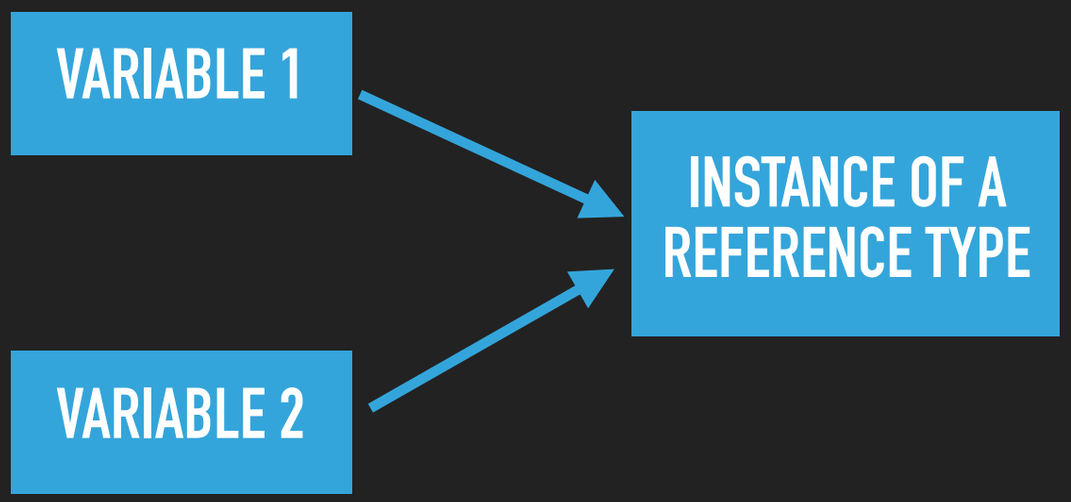
WHAT IS MEMORY MANAGEMENT?

- Value types are kept in memory for the scope of the variable.
- Example: You declare an `Int` variable inside of a function. The memory used to store the integer will be freed* when the function returns.

*YOU CAN ALSO SAY THAT MEMORY WILL BE "DEALLOCATED" OR "RELEASED"

WHAT IS MEMORY MANAGEMENT?

- When can reference types be deallocated?
- Even if a variable goes out of scope, there could be other variables pointing to that instance.



AUTOMATIC REFERENCE COUNTING

- Both Swift and Objective C use ARC (automatic reference counting)
    - ARC keeps track of how many references there are to a given instance
    - Once the reference count goes down to zero, the object can be released from memory

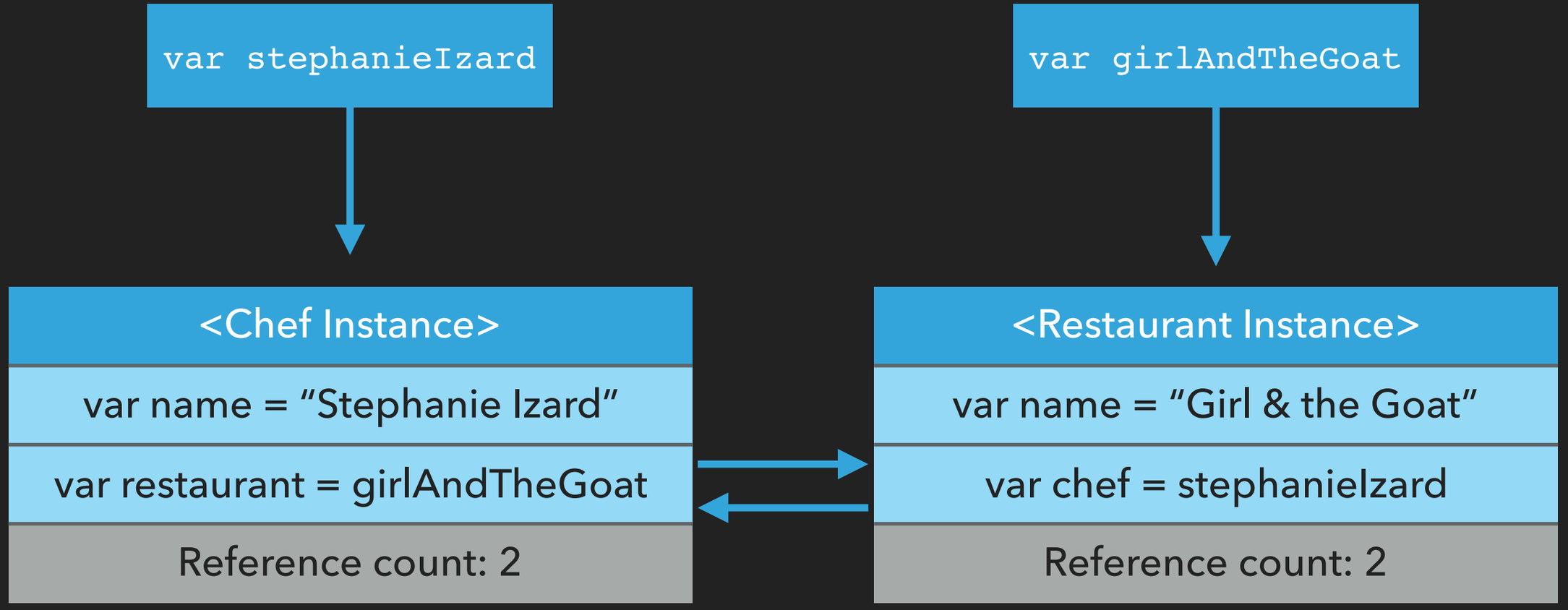
AUTOMATIC REFERENCE COUNTING

- Once there are no more references to an instance, it will be deinitialized (i.e., released from memory)

```
3  class City {
4      init() {
5          print("initializing...")
6      }
7      deinit {
8          print("deinitializing...")
9      }
10 }
11
12 var chicago: City? = City() // prints "initializing..."
13 var windyCity = chicago
14
15 chicago = nil
   windyCity = nil // prints "deinitializing..."
17
```

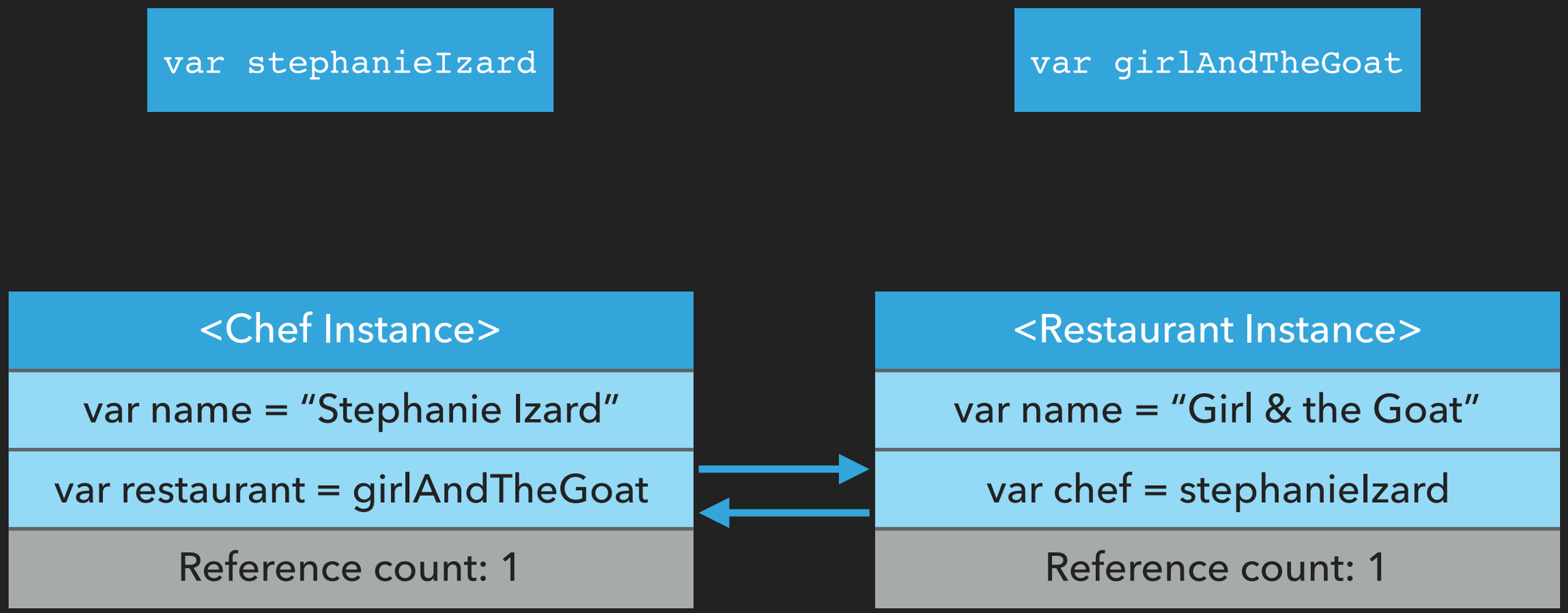
STRONG REFERENCE CYCLES

- If two variables point to the same instance, this creates a strong reference cycle



STRONG REFERENCE CYCLES

- Even after the variables no longer point to the instances, the memory can’t be released



STRONG REFERENCE CYCLES

- Even though we can’t access the chef or restaurant instances, they are still stored in memory

```
33  var stephanieIzard: Chef? = Chef(name: "Stephanie Izard")
34  var girlAndTheGoat: Restaurant? = Restaurant(name: "Girl & the Goat")
35
36  stephanieIzard?.restaurant = girlAndTheGoat
37  girlAndTheGoat?.chef = stephanieIzard
38
39  stephanieIzard = nil
40  girlAndTheGoat = nil
41
```

STRONG REFERENCE CYCLES

- This creates a memory leak

```
33 var stephanieIzard: Chef? = Chef(name: "Stephanie Izard")
34 var girlAndTheGoat: Restaurant? = Restaurant(name: "Girl & the Goat")
35
36 stephanieIzard?.restaurant = girlAndTheGoat
37 girlAndTheGoat?.chef = stephanieIzard
38
39 stephanieIzard = nil
40 girlAndTheGoat = nil
41
```

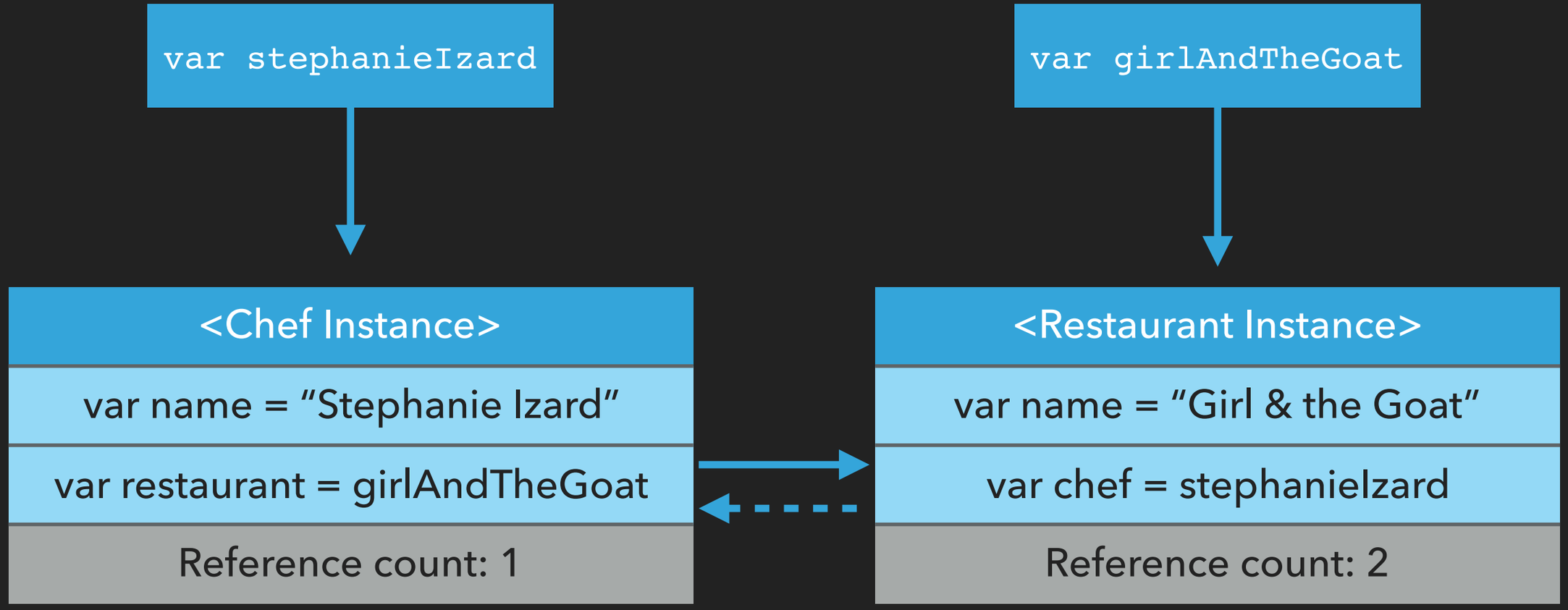
FIXING STRONG REFERENCE CYCLES

- You can mark a variable as `weak` to break a strong reference cycle

```
19  class Restaurant {
20      let name: String
21      weak var chef: Chef? // <-- change this variable to weak
22
23      init(name: String) {
24          self.name = name
25          print("initializing \(name)")
26      }
27
28      deinit {
29          print("deinitializing \(name)")
30      }
31  }
```

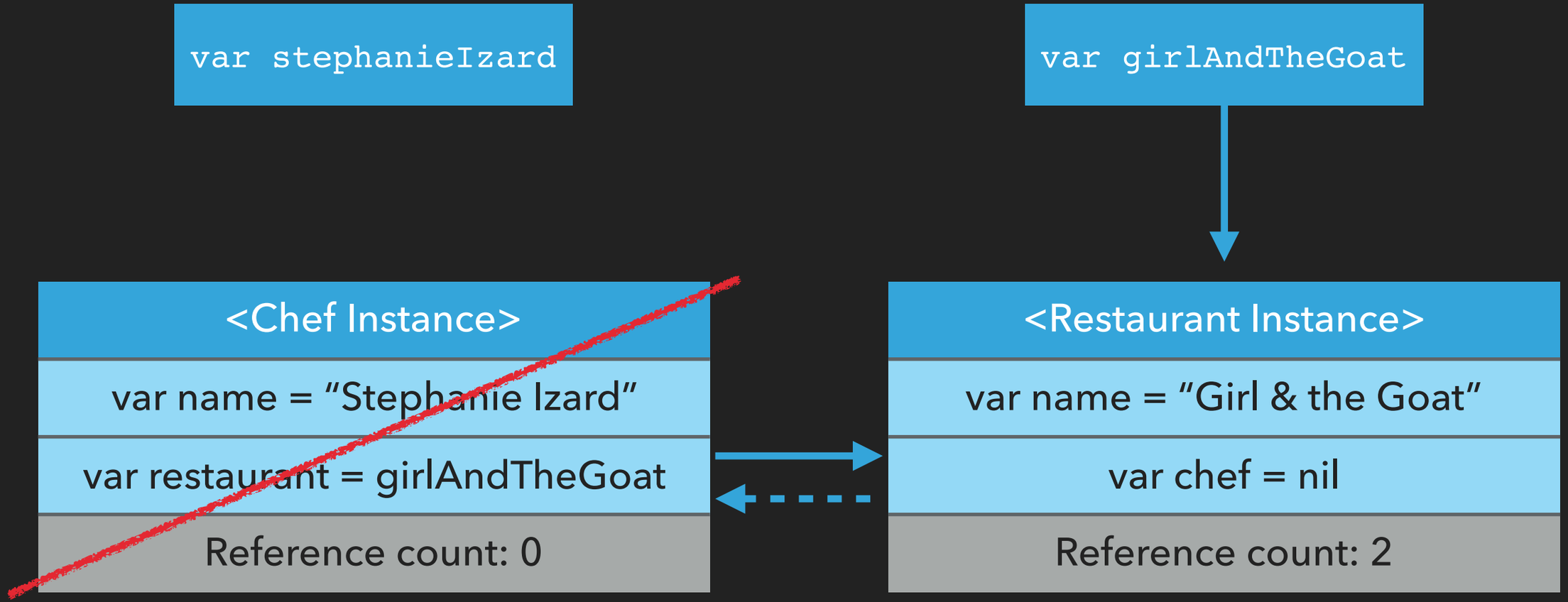
FIXING STRONG REFERENCE CYCLES

- The `weak` keyword indicates that it shouldn't count towards the instance's reference count



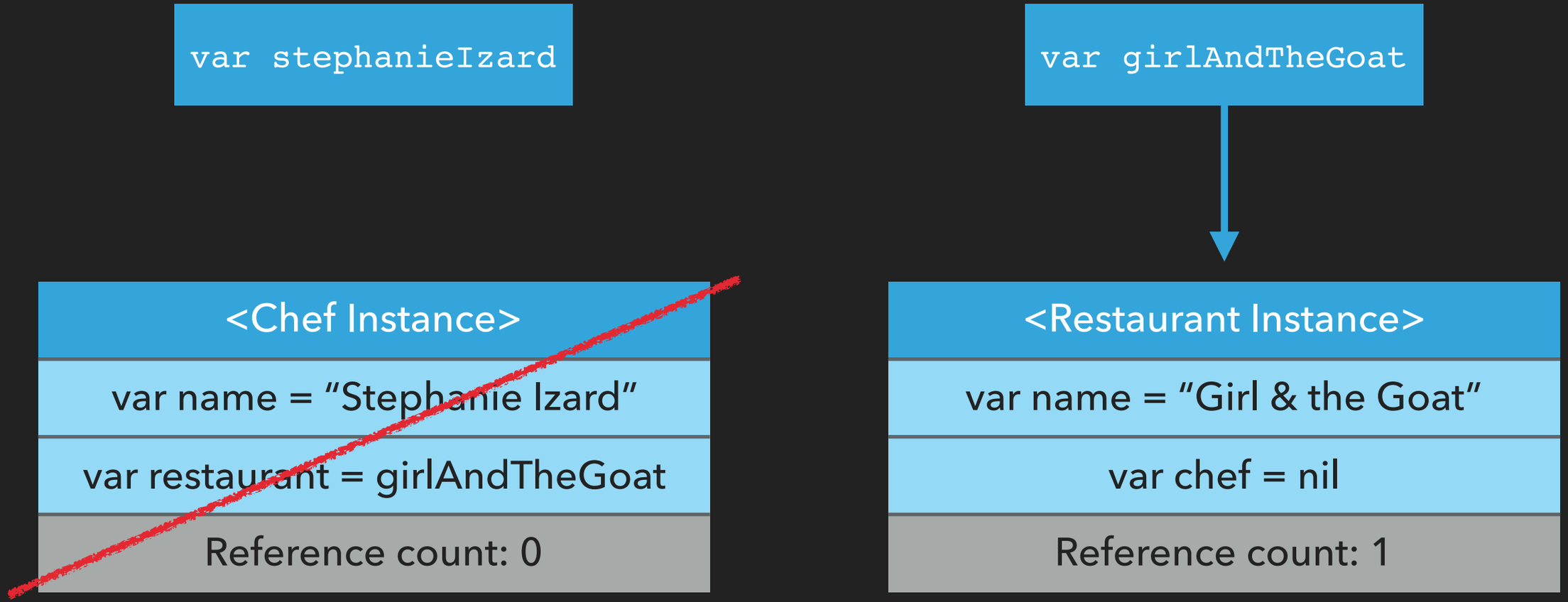
FIXING STRONG REFERENCE CYCLES

- When the `stephanieIzard` variable is set to `nil`, the chef instance's reference count goes down to zero



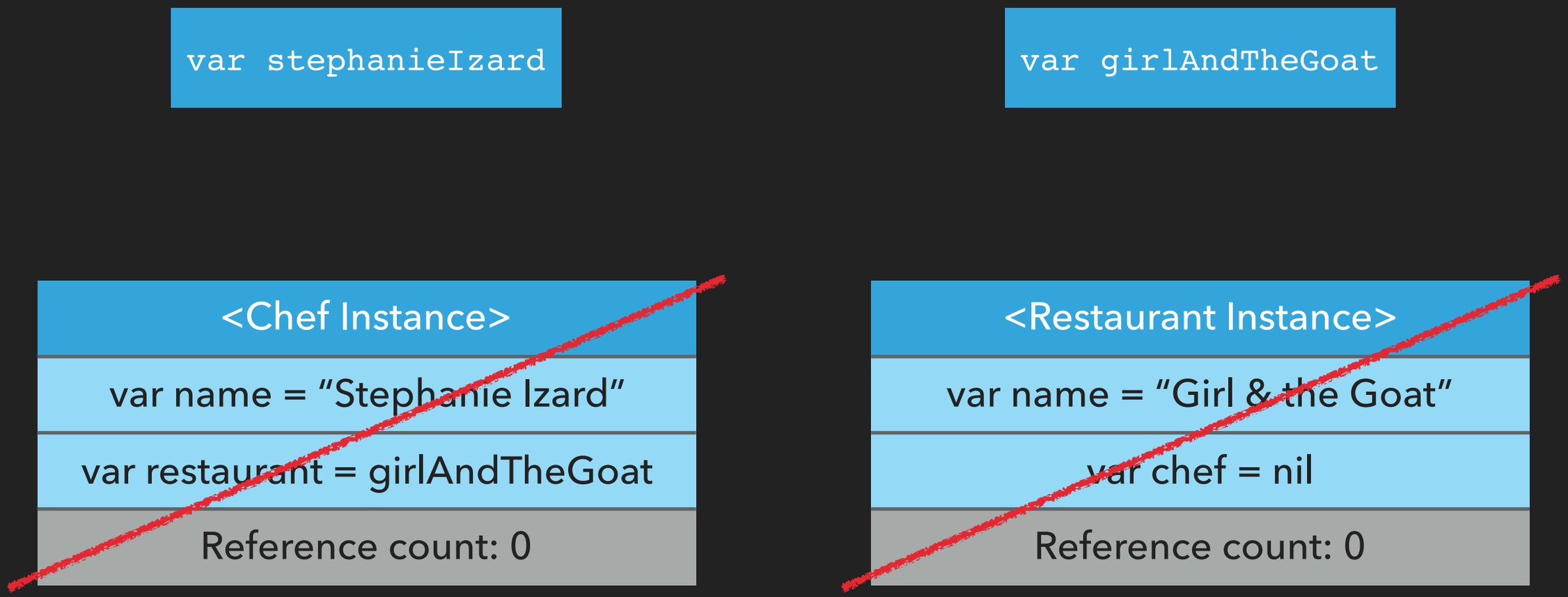
FIXING STRONG REFERENCE CYCLES

- This causes the restaurant instance's reference count to go down to one



FIXING STRONG REFERENCE CYCLES

- Once the `girlAndTheGoat` variable is set to nil, the restaurant instance can be deallocated



BUILDING A

CUSTOM CONTROL

BEYOND TABLE VIEWS…

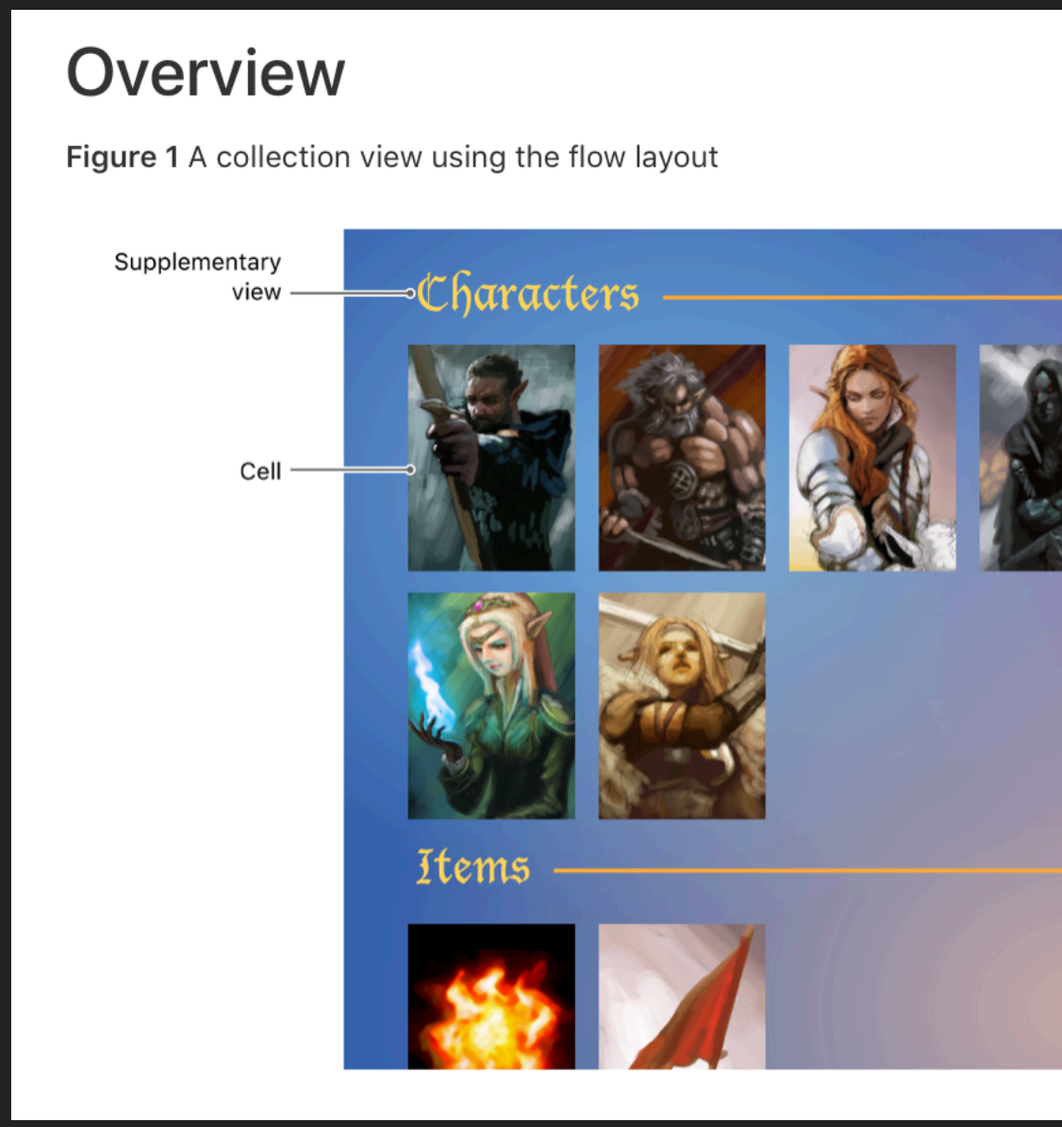
UICOLLECTIONVIEW

LIMITATIONS OF TABLE VIEWS

- Only allows rows
- Only allows vertical scrolling

INTRODUCING COLLECTION VIEWS

- Rows and columns
- Scrolling can be vertical or horizontal
- You can create much more complex layouts



https://developer.apple.com/documentation/uikit/uicollectionview

COMPARISON WITH TABLE VIEWS

UITableView	UICollectionView
UITableViewCell	UICollectionViewCell
UITableViewDataSource	UICollectionViewDataSource
UITableViewDelegate	UICollectionViewDelegate
UITableViewController	UICollectionViewController

DATASOURCE

- `UICollectionViewDataSource` methods

Getting Item and Section Metrics

```
func collectionView(UICollectionView, numberOfItemsInSection: Int)
-> Int
```

Asks your data source object for the number of items in the specified section.

Required.

Note the use of “item” instead of “row”

```
func numberOfSections(in: UICollectionView) -> Int
```

Asks your data source object for the number of sections in the collection view.

Getting Views for Items

```
func collectionView(UICollectionView, cellForItemAt: IndexPath) ->
UICollectionViewCell
```

Asks your data source object for the cell that corresponds to the specified item in the collection view.

Required.

DELEGATE

- `UICollectionViewDelegate` methods

Managing the Selected Cells

```
func collectionView(UICollectionView, shouldSelectItemAt: IndexPath) -> Bool
```

Asks the delegate if the specified item should be selected.

```
func collectionView(UICollectionView, didSelectItemAt: IndexPath)
```

Tells the delegate that the item at the specified index path was selected.

```
func collectionView(UICollectionView, shouldDeselectItemAt: IndexPath) -> Bool
```

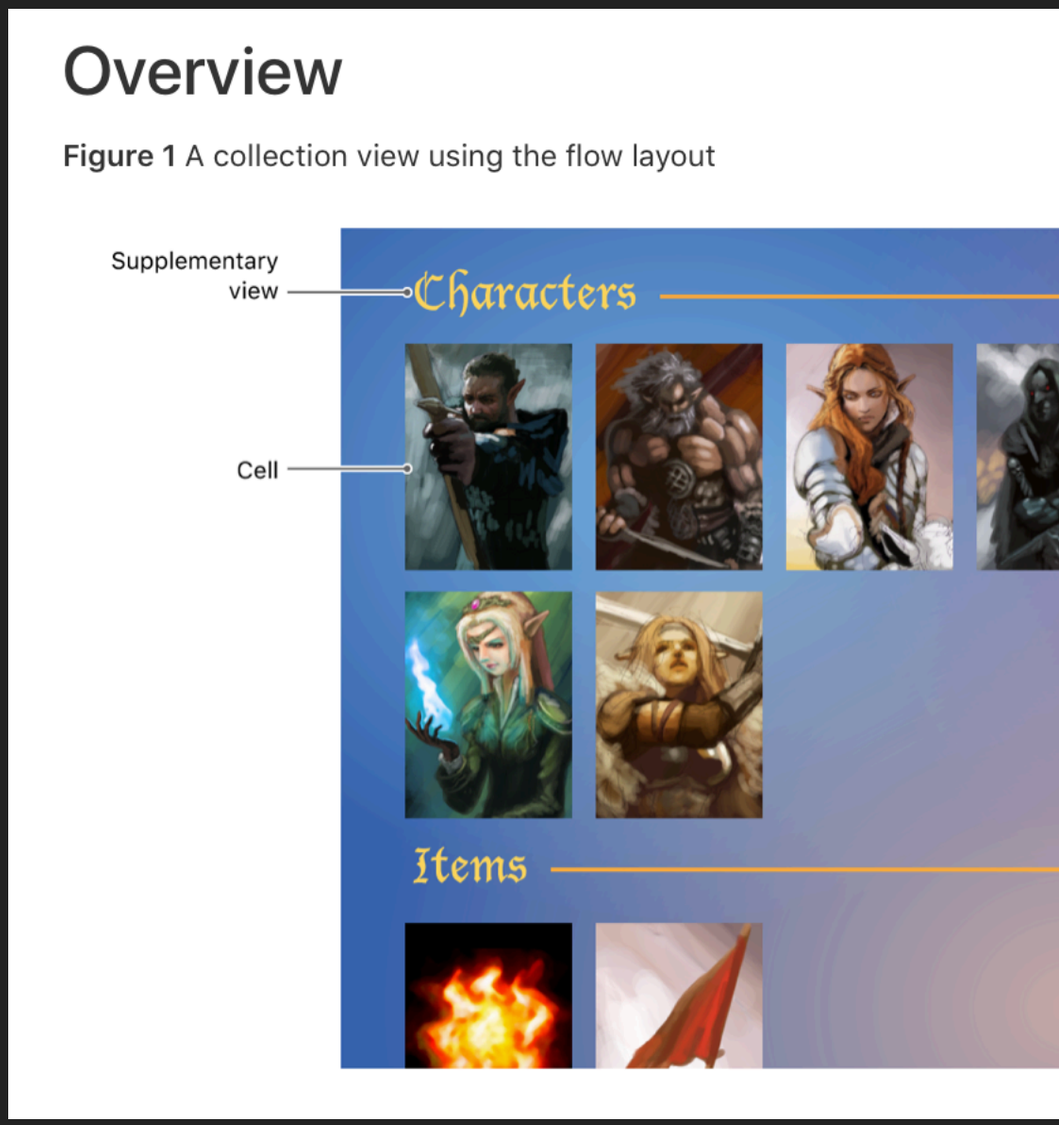
Asks the delegate if the specified item should be deselected.

```
func collectionView(UICollectionView, didDeselectItemAt: IndexPath)
```

Tells the delegate that the item at the specified path was deselected.

USING COLLECTION VIEWS

- In addition to cells, collection views can have supplementary views
- Used to create headers and footers for different sections



https://developer.apple.com/documentation/uikit/uicollectionview

COLLECTION VIEW LAYOUT

- Unlike table views, collection views must be given a layout object
- This determines the placement of cells and supplementary views
- `UICollectionViewLayout` is the base class for any layout

COLLECTION VIEW LAYOUT

- Before iOS 13, there were two options to define a layout
  - Option 1: `UICollectionViewFlowLayout`
  - Option 2: Create a custom subclass of `UICollectionViewLayout`

COLLECTION VIEW LAYOUT

- `UICollectionViewFlowLayout`
  - Organizes items into a grid with optional header and footer views for each section
- Limited options for layout, but easy to use

COLLECTION VIEW LAYOUT

- Custom subclass of `UICollectionViewLayout`
    - Allows you to define the exact placement of each item and supplementary view
- Infinite options for layout, but more difficult to use

COLLECTION VIEW LAYOUT

- iOS 13 introduce a third layout option called `UICollectionViewCompositionalLayout`

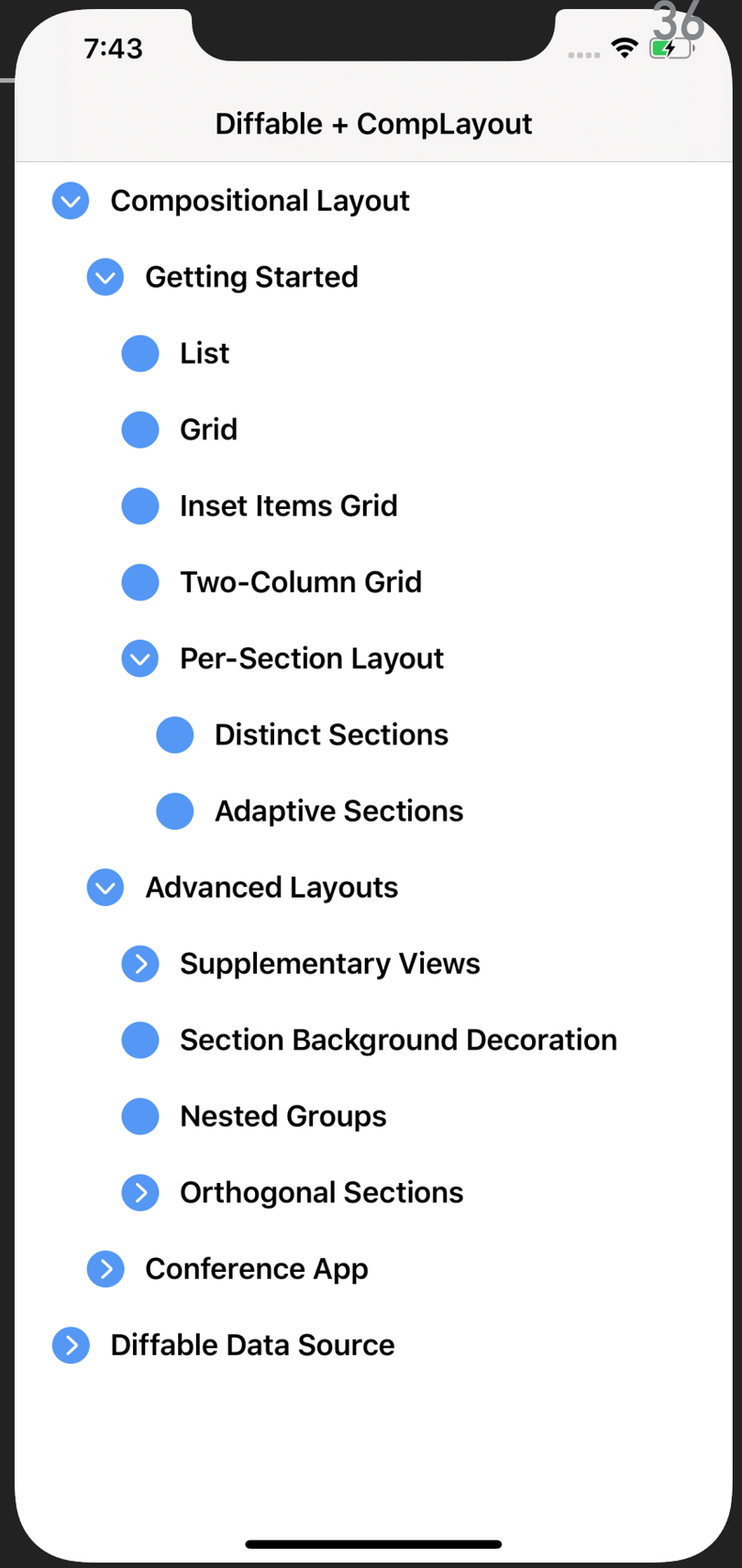
COLLECTION VIEW

COMPOSITIONAL LAYOUT

COMPOSITIONAL LAYOUT

- Composable - make one complex thing from lots of simple things
- Flexible - design almost any layout
- Fast - framework handles performance considerations

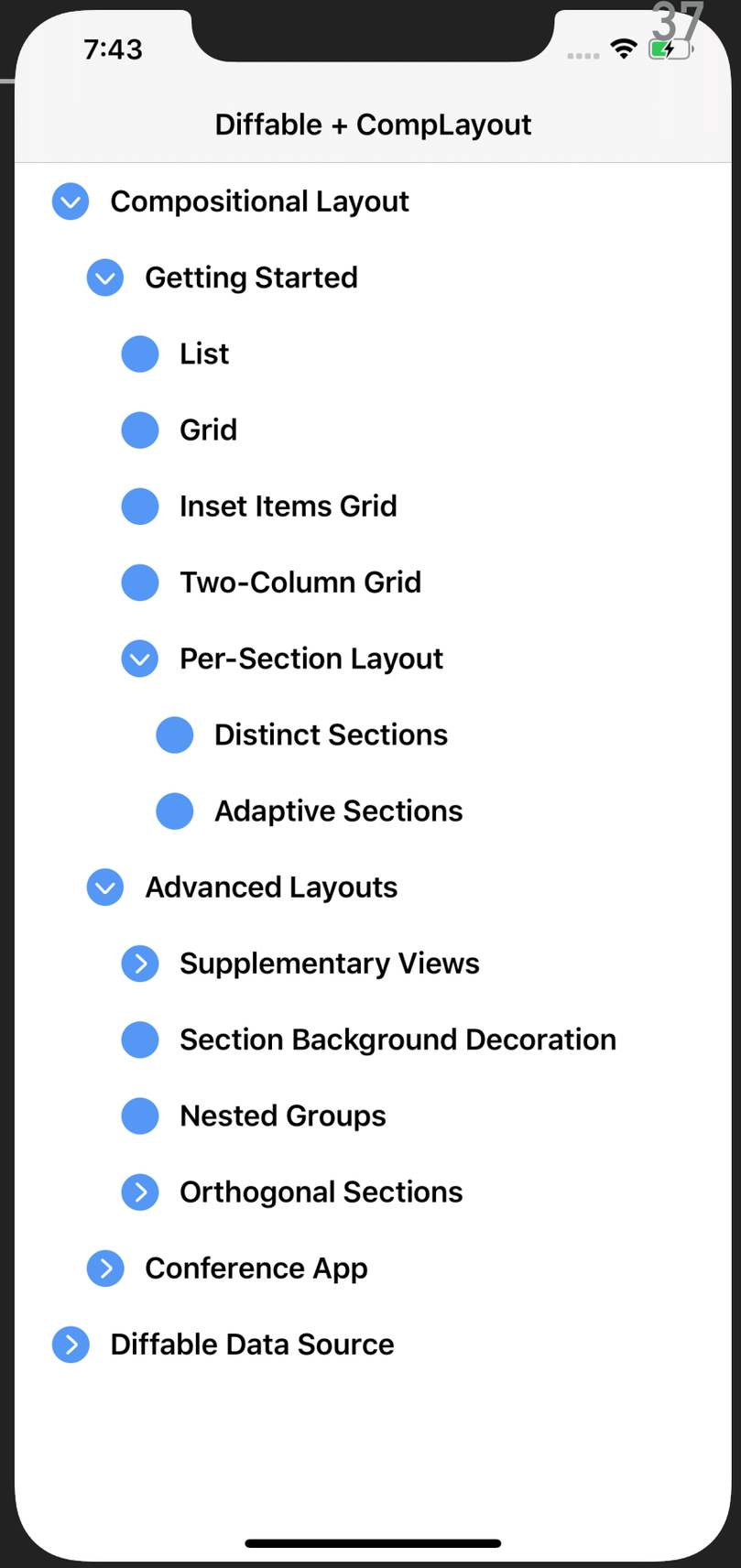
WWDC Demo App



COMPOSITIONAL LAYOUT

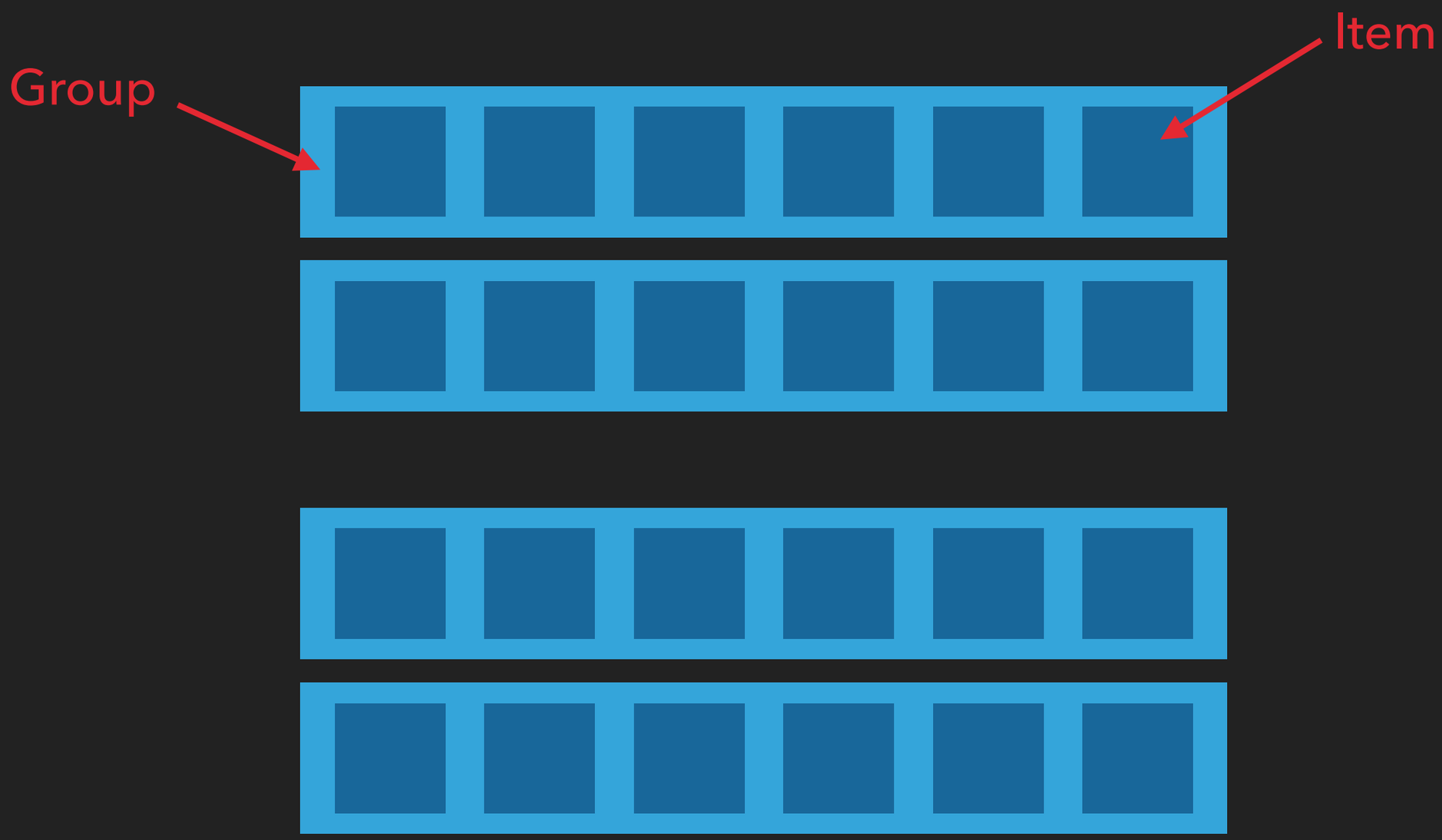
- Four key components:
  - Item
  - Group
  - Section
  - Layout

WWDC Demo App

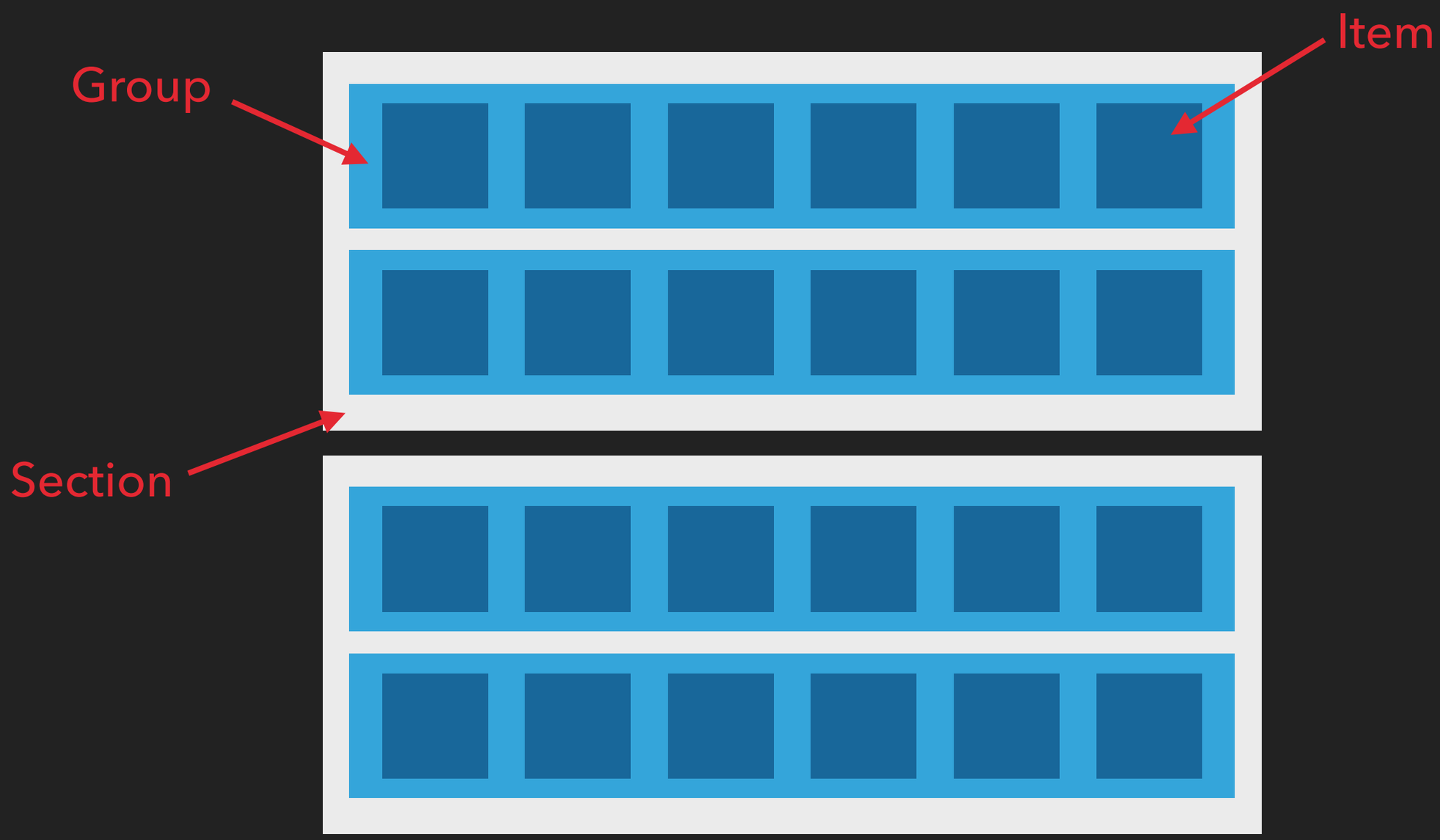


COMPOSITIONAL LAYOUT

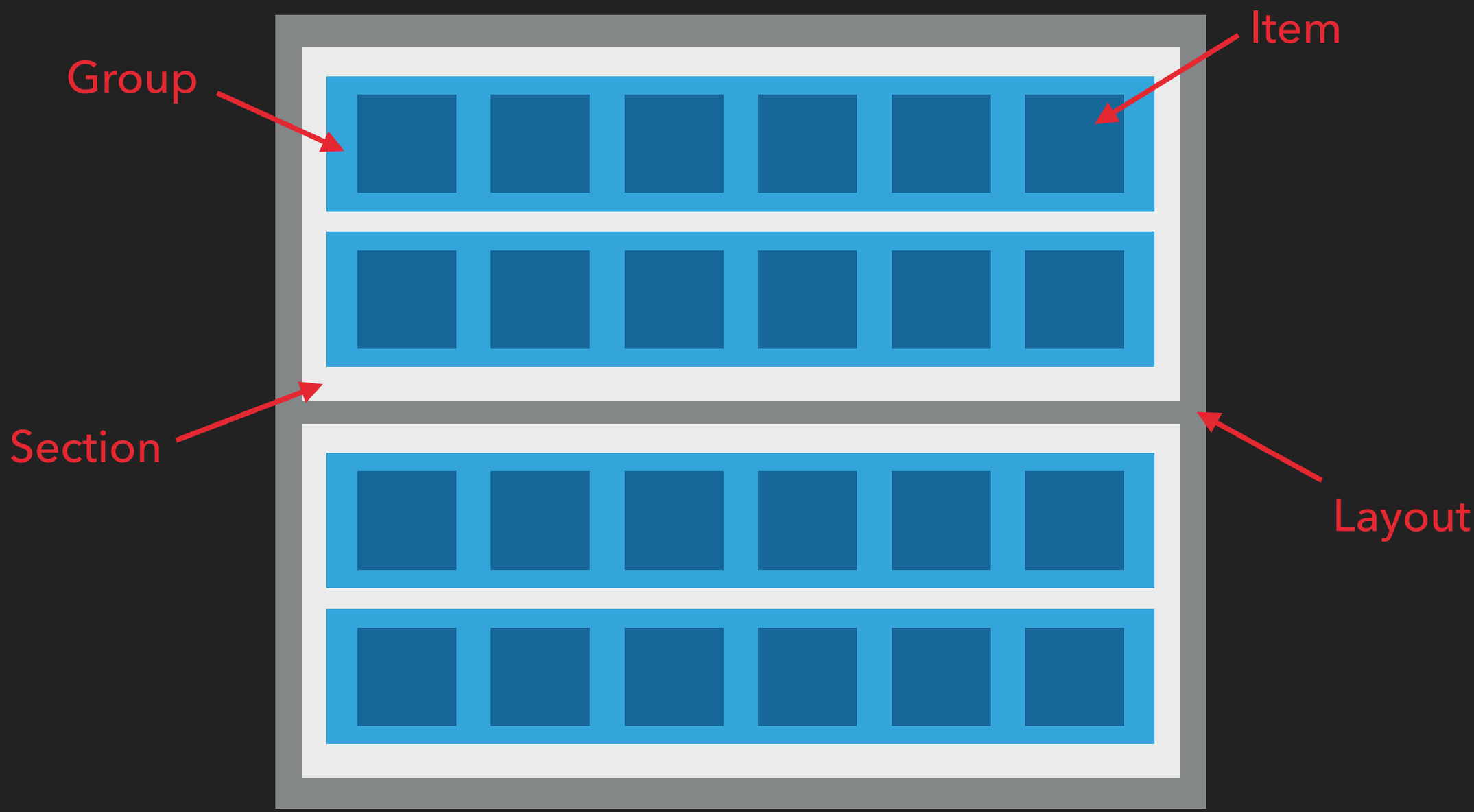
COMPOSITIONAL LAYOUT



COMPOSITIONAL LAYOUT



COMPOSITIONAL LAYOUT



COMPOSITIONAL LAYOUT

- Start by creating an item with a layout size

```
let itemSize = NSCollectionLayoutSize(widthDimension: .fractionalWidth(0.5),
                                      heightDimension: .fractionalHeight(1.0))

let item = NSCollectionLayoutItem(layoutSize: itemSize)
```

COMPOSITIONAL LAYOUT

- Width and height dimensions can be expressed as fractions of the item's container

```
let itemSize = NSCollectionLayoutSize(widthDimension: .fractionalWidth(0.5),
                                      heightDimension: .fractionalHeight(1.0))

let item = NSCollectionLayoutItem(layoutSize: itemSize)
```

COMPOSITIONAL LAYOUT

- Alternatively, width and height dimensions can be expressed as absolutes

```
let itemSize = NSCollectionLayoutSize(widthDimension: .absolute(100),
                                      heightDimension: .absolute(30))

let item = NSCollectionLayoutItem(layoutSize: itemSize)
```

COMPOSITIONAL LAYOUT

- Create a group with a layout size and an array of subitems

```
let groupSize = NSCollectionLayoutSize(widthDimension: .fractionalWidth(1.0),
                                       heightDimension: .absolute(44))

let group = NSCollectionLayoutGroup.horizontal(layoutSize: groupSize,
                                               subitems: [item])
```

COMPOSITIONAL LAYOUT

- Create a section with a group

```
let section = NSCollectionLayoutSection(group: group)
```

COMPOSITIONAL LAYOUT

- Create a layout with a section

```
let layout = UICollectionViewCompositionalLayout(section: section)
```

COMPOSITIONAL LAYOUT

- Set the layout on the collection view

```
collectionView.setCollectionViewLayout(layout, animated: false)
```

COLLECTION VIEW

DIFFABLE DATASOURCE

DIFFABLE DATASOURCE

- Sometimes, we need to make updates to the data displayed in a table or collection view
  - Inserts
  - Deletes
  - Reorders

WWDC Demo App

DIFFABLE DATASOURCE

- Before iOS 13, changes could be made using the `performBatchUpdates` method
    - First, update the array that holds the data for your table or collection view
    - Second, tell the table or collection view to insert/delete/move certain items

DIFFABLE DATASOURCE

- If the UI and data somehow got out of sync, the app would crash

```
*** Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason:
'Invalid update: invalid number of sections.  The number of sections contained in the
collection view after the update (10) must be equal to the number of sections contained in the
collection view before the update (10), plus or minus the number of sections inserted or
deleted (0 inserted, 1 deleted).'
***
```

DIFFABLE DATASOURCE

- To avoid this, developers would often call `reloadData` on the entire table or collection view
    - This isn't very efficient, since you reload everything, not just what changed
- Users don't see the changes animate

DIFFABLE DATASOURCE

- In iOS 13, Apple introduce diffable datasource
  - `UITableViewDiffableDataSource`
  - `UICollectionViewDiffableDataSource`

DIFFABLE DATASOURCE

- Instead of trying to keep your UI and data in sync, you provide snapshots to the diffable datasource
- The diffable datasource figures out what's changed and animates the changes

DIFFABLE DATASOURCE

- UICollectionViewDiffableDataSource is a generic type
  - Concrete types for section and item need to be specified
- Section and item must be hashable

```
var dataSource: UICollectionViewDataSource<Int, String>
```

Section

Item

DIFFABLE DATASOURCE

- UICollectionViewDiffableDataSource is initialized with two parameters
    - An instance of UICollectionView
    - A closure that can configure each item in the collection view

DIFFABLE DATASOURCE

- Create a snapshot with the section and item data
- Apply the snapshot to the datasource to display the data
- The datasource will animate any changes to the snapshot by default

CONCURRENCY WITH…

GRAND CENTRAL DISPATCH

MULTITASKING

- Most apps will need to do more than one thing at a time.

https://talkroute.com/multitasking-vs-single-tasking-which-is-more-effective/

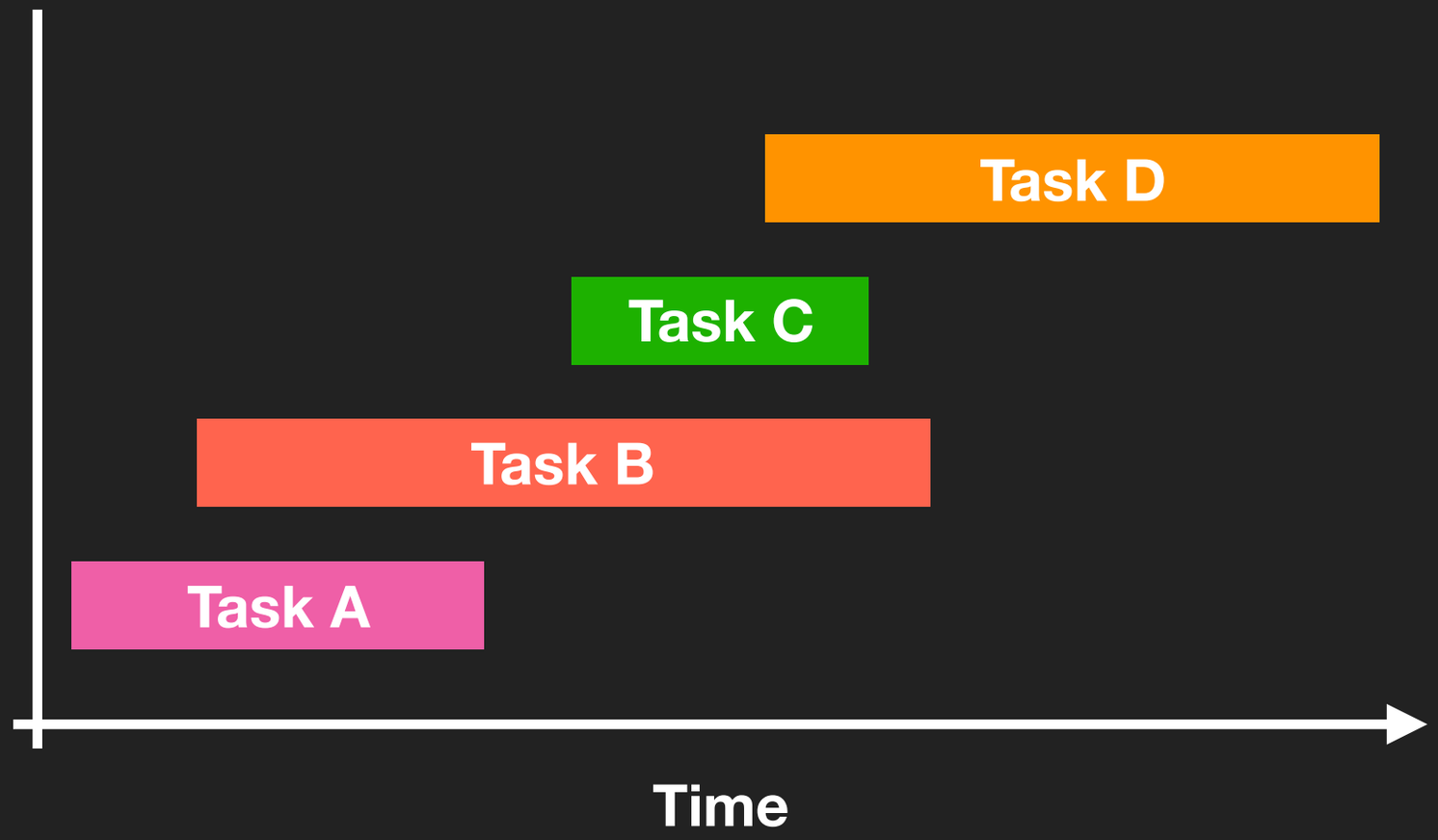
MULTITASKING

- For example:
  - Scroll table view as the user swipes
  - Fetch more data to display in the table view

https://talkroute.com/multitasking-vs-single-tasking-which-is-more-effective/

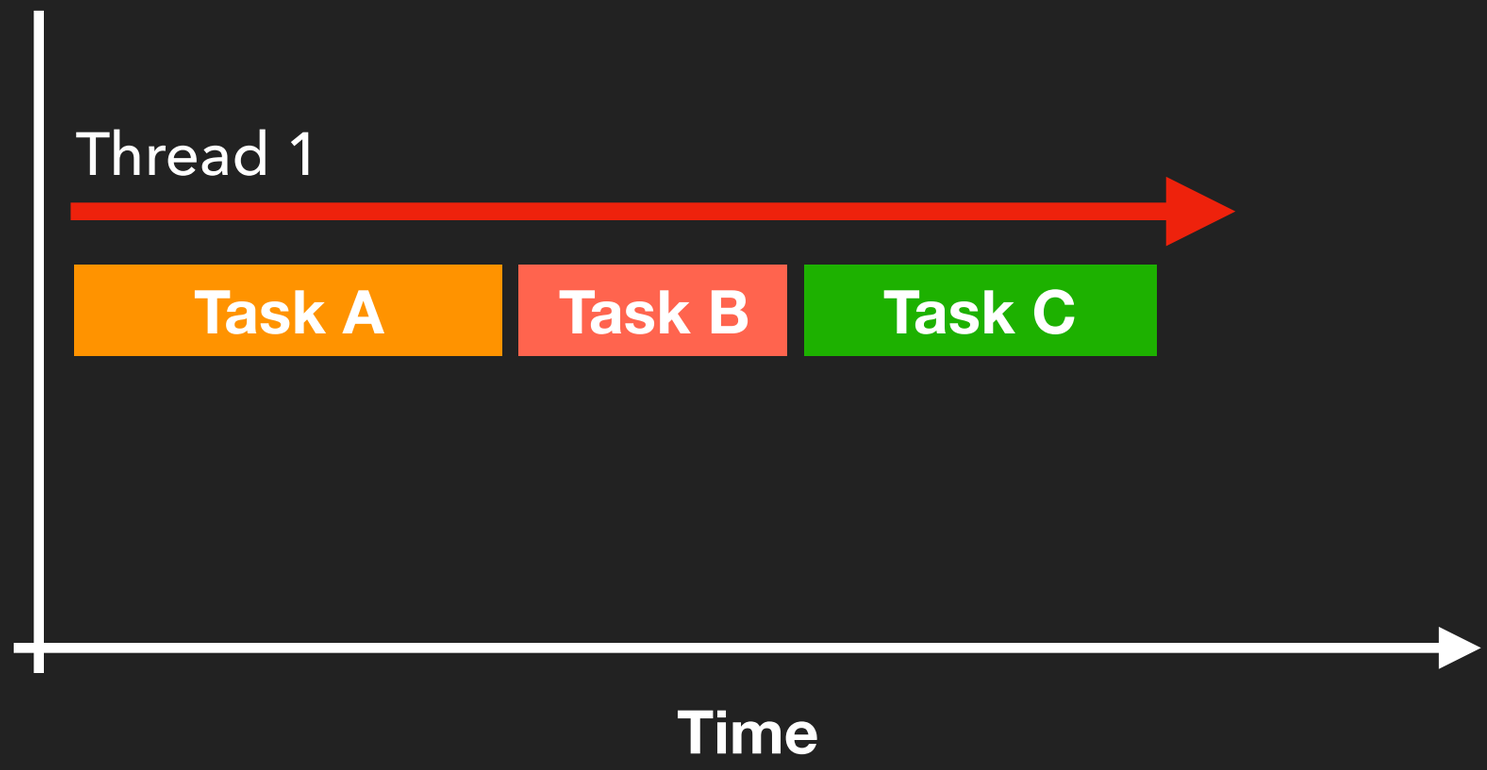
MULTITASKING

- **Concurrency** is when multiple blocks of code execute at the same time



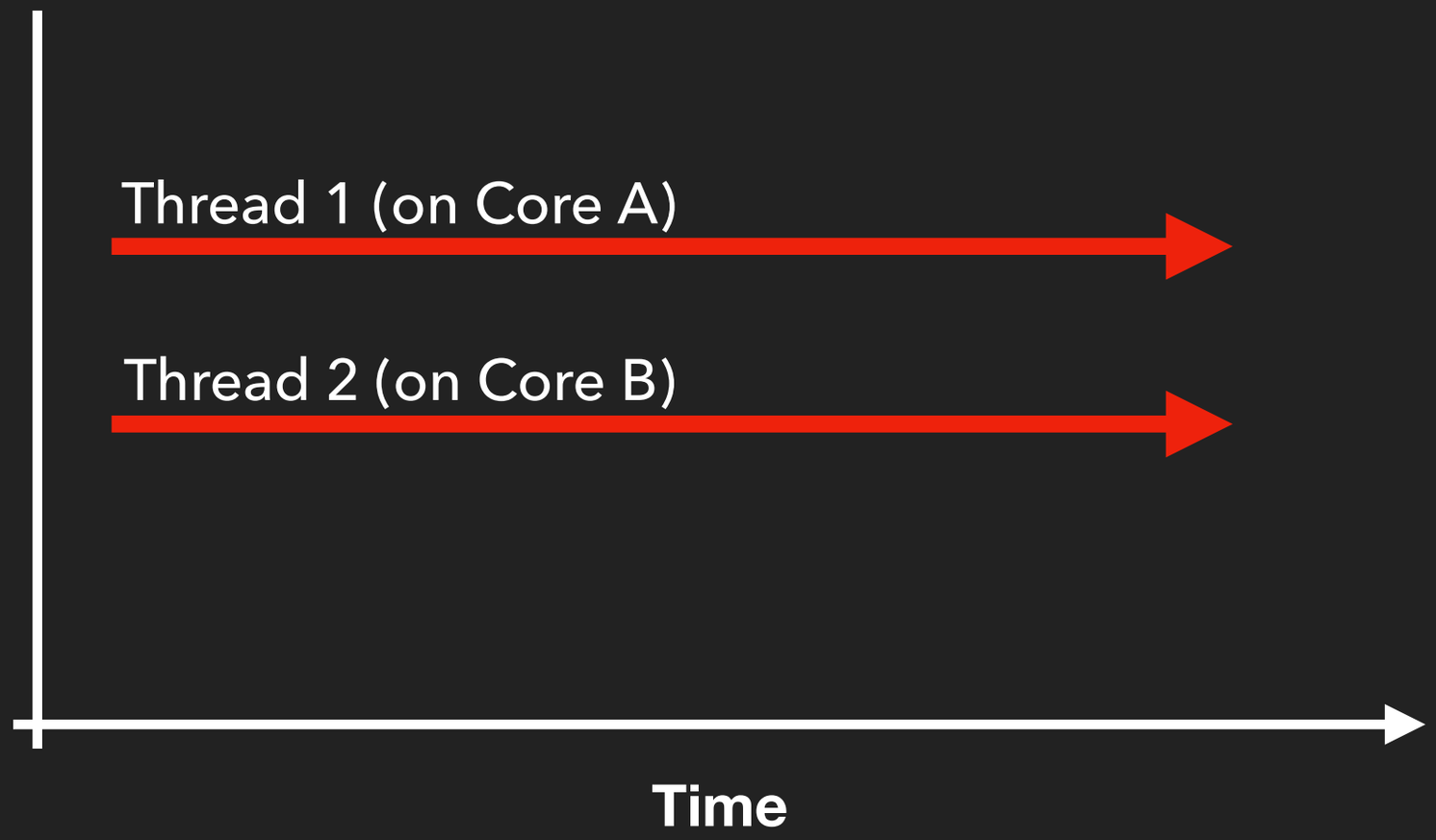
MULTITASKING

- Each block of code (or **task**) executes on a **thread**
  - A thread can only execute one task at a time



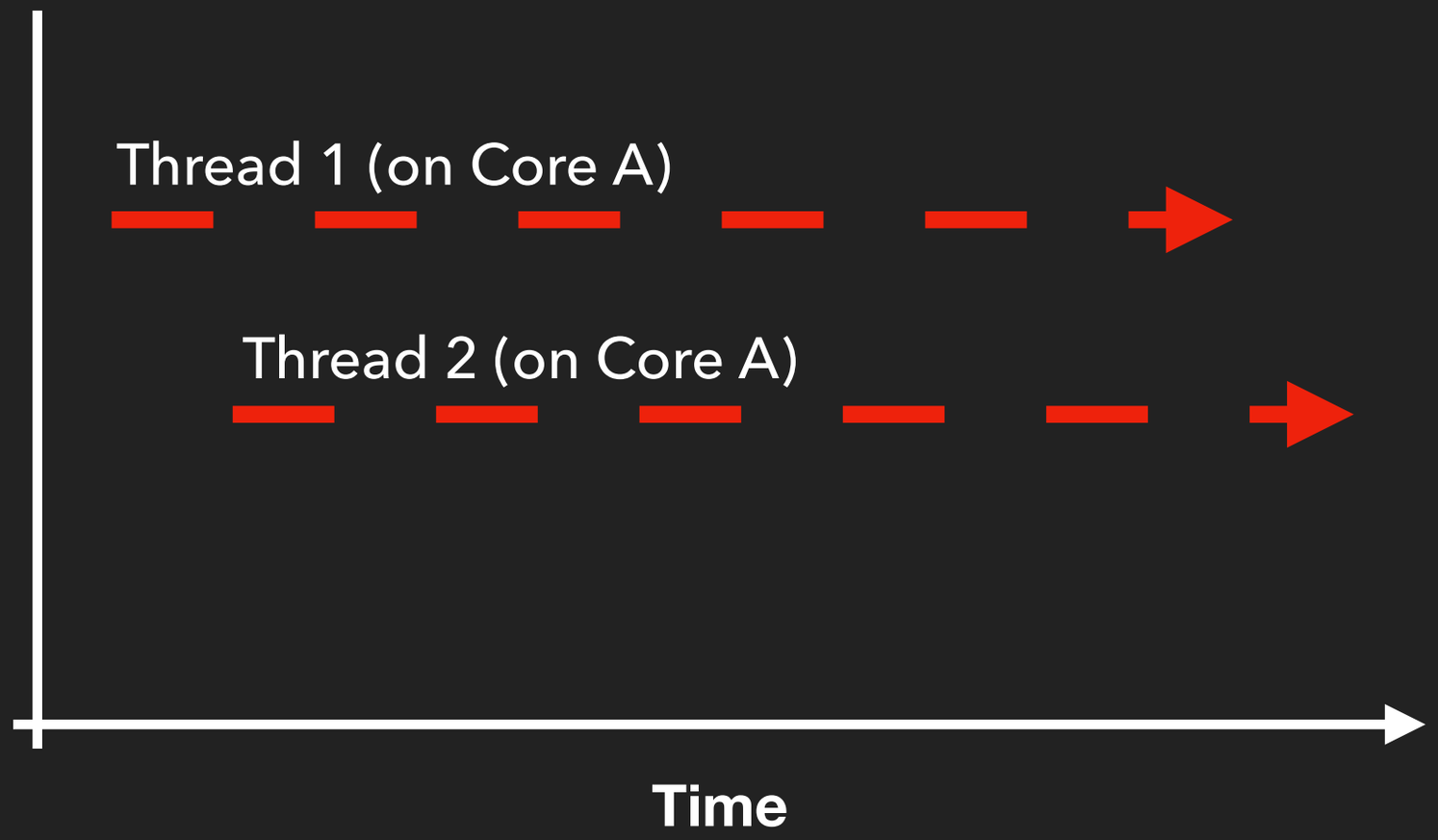
MULTITASKING

- **Concurrency** can be achieved by running threads on multiple cores…



MULTITASKING

- … or by context-switching between threads on a single core

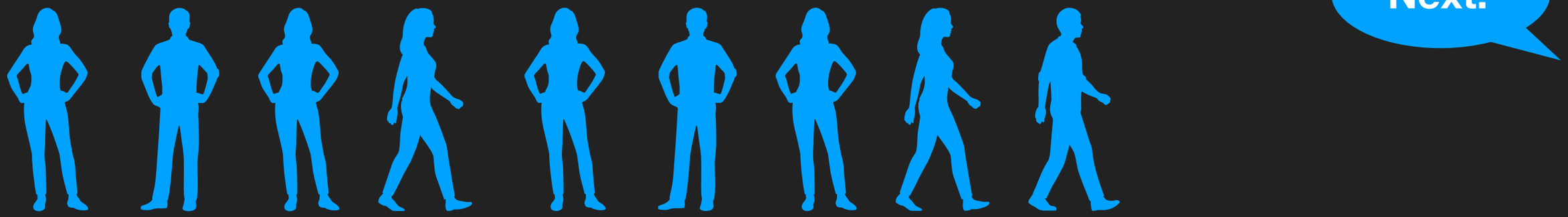


MULTITASKING

- **Grand Central Dispatch** is the system in iOS that manages threads for you
  - You put a block of code on a **dispatch queue**
  - Each queue corresponds to one or more threads
  - GCD decides which thread will run your block of code

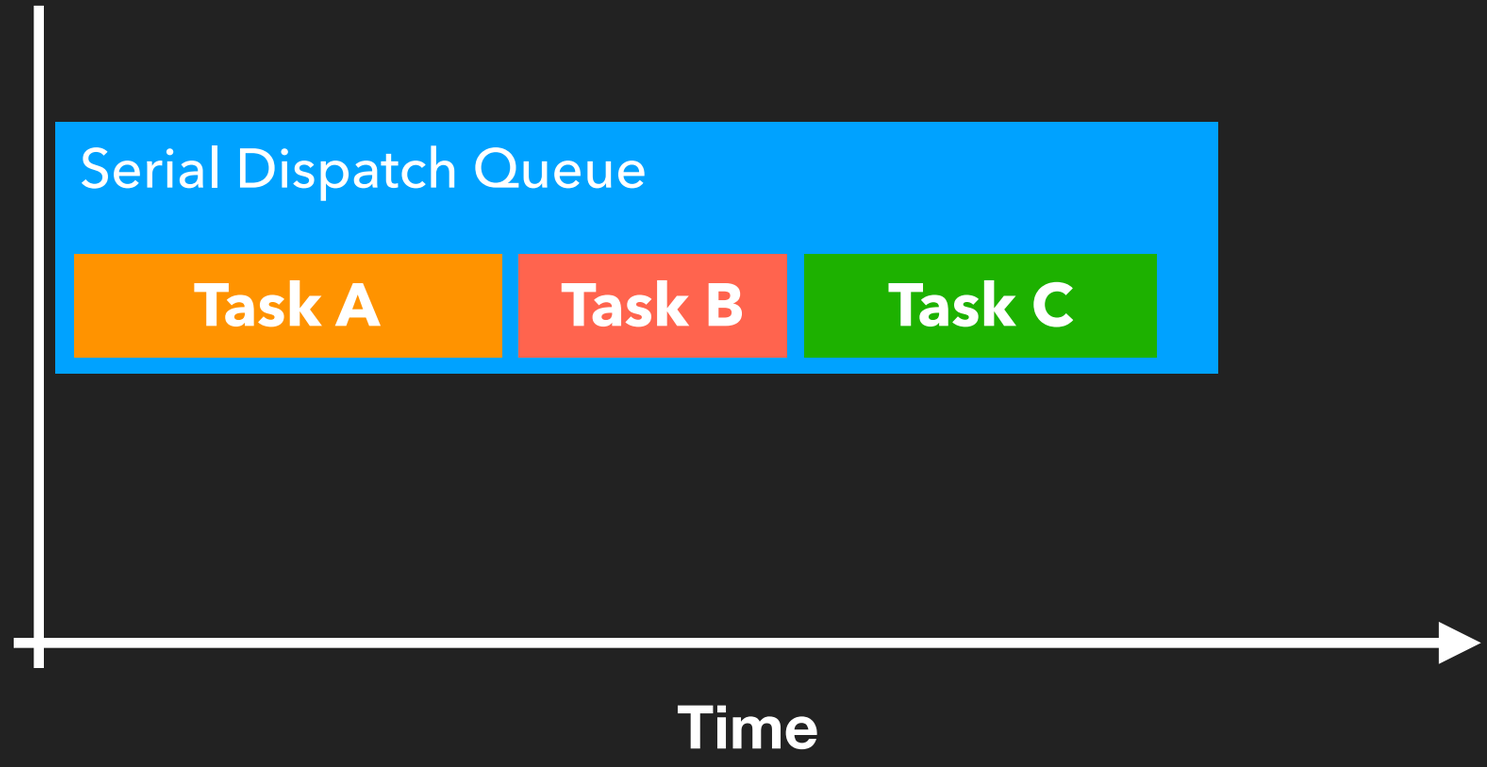
DISPATCH QUEUES

- Dispatch queues execute blocks of code in FIFO (first-in, first-out) order



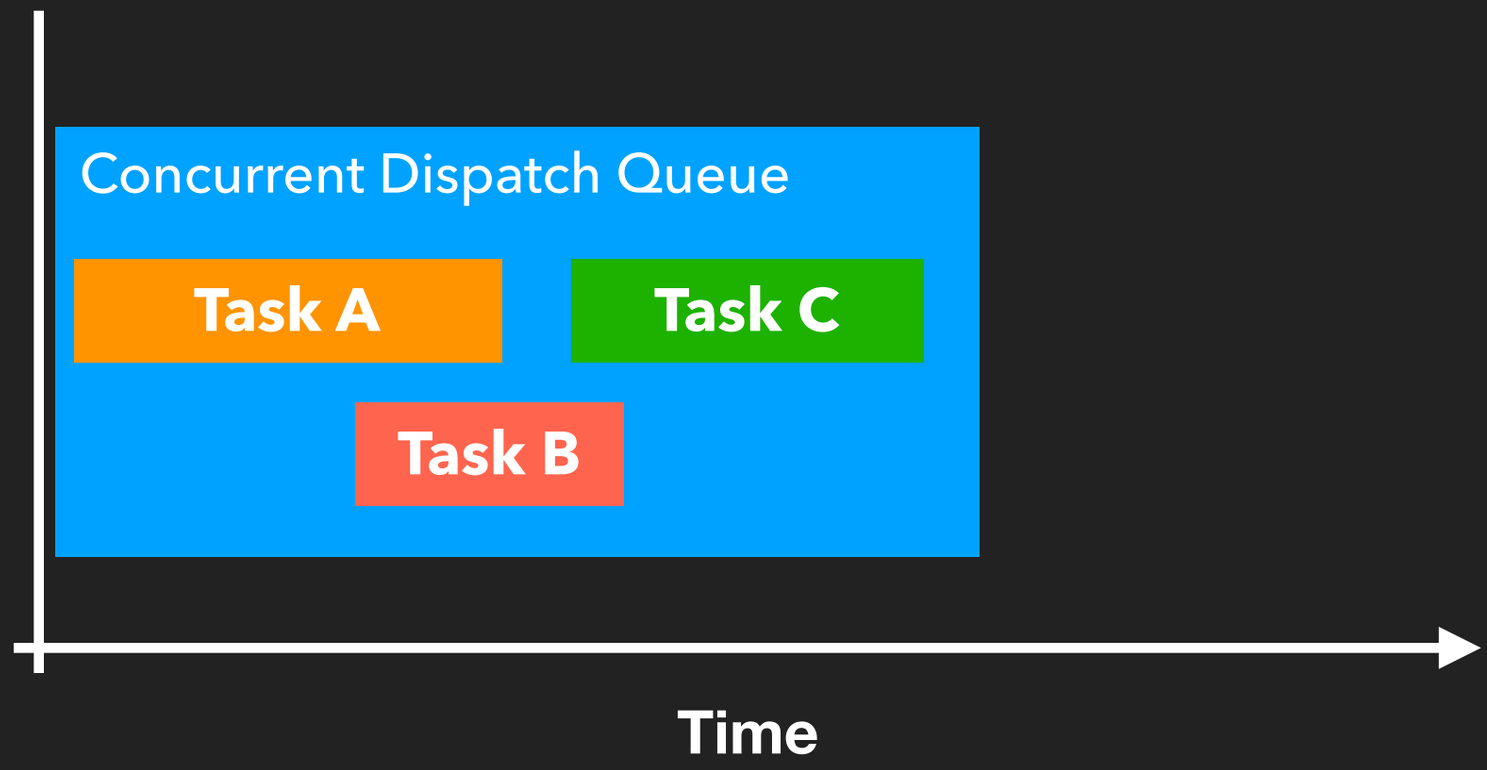
DISPATCH QUEUES

- Queues can be **serial**



DISPATCH QUEUES

- Or they can be **concurrent**



QUALITY OF SERVICE

- GCD prioritizes queues using a quality of service property

```
enum QualityOfService {
    case userInteractive
    case userInitiated
    case utility
    case background
    case default
}
```

QUALITY OF SERVICE

- `userInteractive`: used for user interface updates and event handling
    - Updating the text of a label
    - Scrolling a table view

QUALITY OF SERVICE

- `userInitiated`: used for tasks explicitly requested by the user that must be completed before interaction continues
  - Loading an email
  - Displaying a web page

QUALITY OF SERVICE

- `utility`: used for tasks for which the user does not expect immediate results, but is typically kept informed of the progress
  - Downloading a podcast
  - Uploading a file to a server

QUALITY OF SERVICE

- `background`: used for tasks not visible to the user
    - Pre-fetching email subject-lines
    - Sending analytics events

QUALITY OF SERVICE

- `default`: allows the system to choose the quality of service

SYNCHRONOUS VS ASYNCHRONOUS

- **Synchronous** means control returns to the call site after the function completes
  - "Execute this task! I'll wait until you're done."
- **Asynchronous** means control returns to the call site immediately
  - "Execute this task! I don't care when it finishes."

TYPES OF DISPATCH QUEUES

	Serial or Concurrent?	Quality of Service	Sync or Async?
`Main`	Serial	User Interactive	Either
`Global`	Concurrent	Any	Either
`Custom`	Either	Any	Either

MAIN DISPATCH QUEUE

- To put code on the main queue, use `DispatchQueue.main.async`

```
DispatchQueue.main.async {
    completion(UIImage(data: data), nil)
}
```

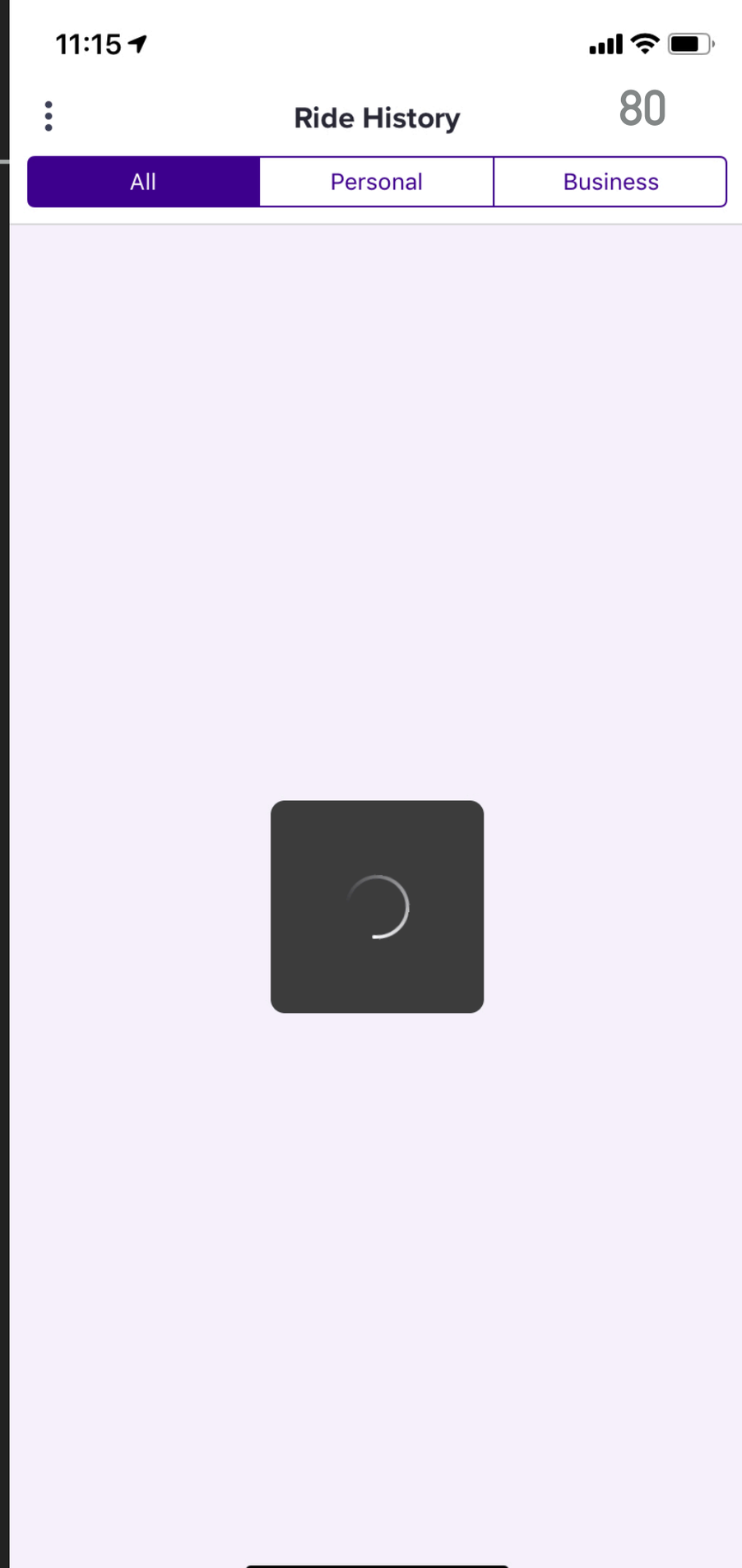
This should look familiar from the networking code used in Assignment 3

BEST PRACTICES FOR

NETWORKING

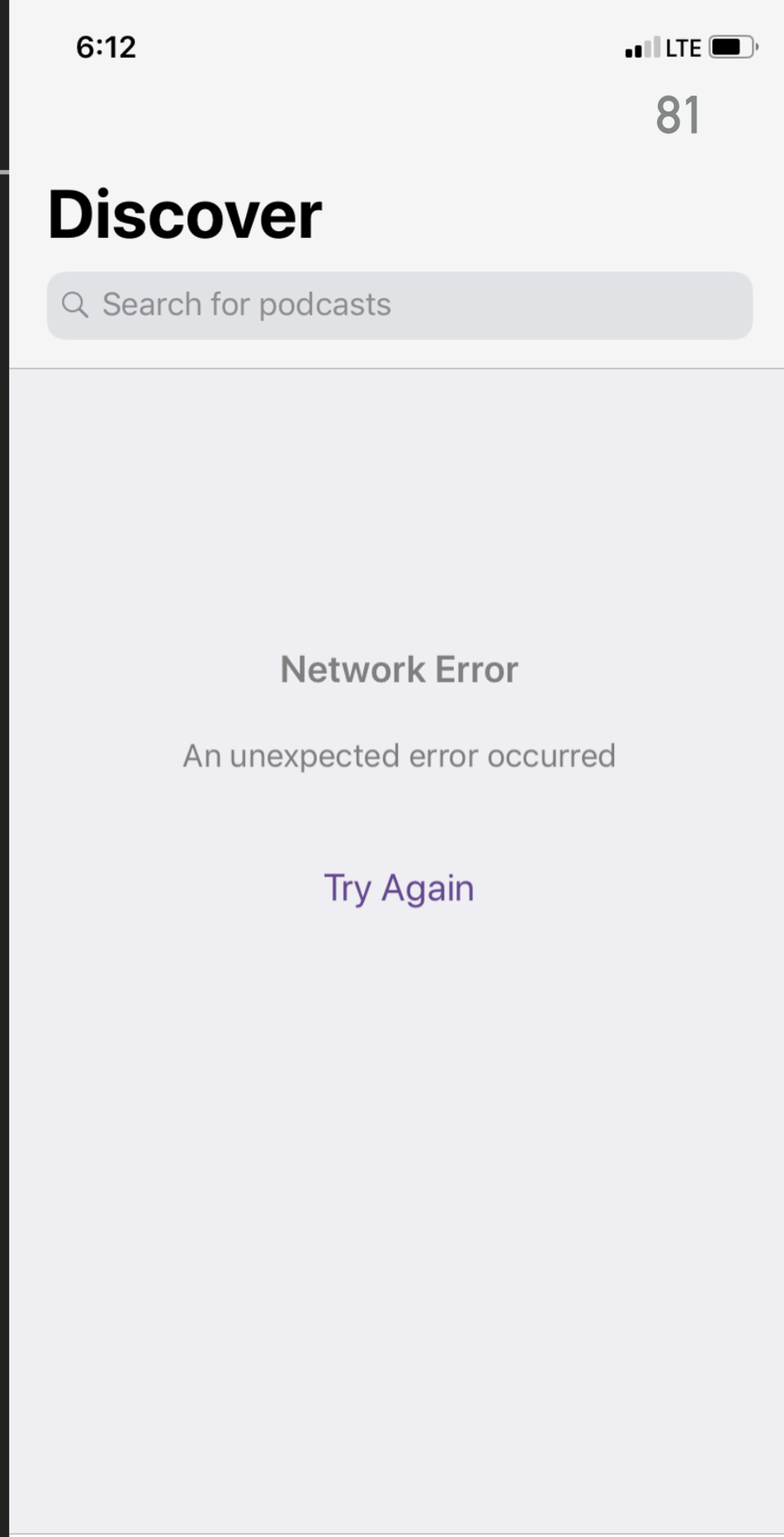
BEST PRACTICE #1

- Show an activity indicator or progress bar when a network request is in progress



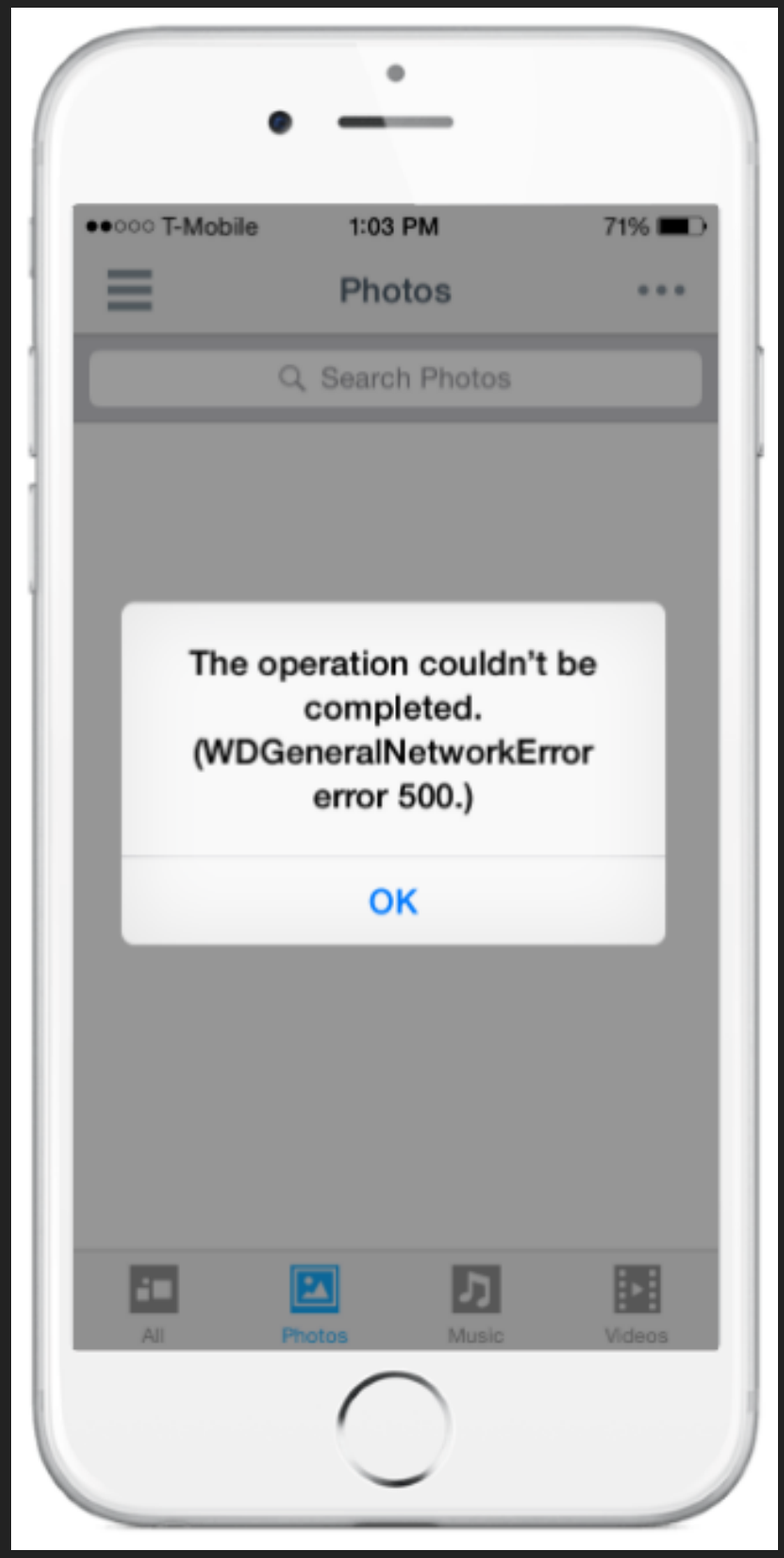
BEST PRACTICE #2

- Notify the user when an error occurs
  - No network connection
  - Server error
  - Unable to parse data
  - Empty result set



BEST PRACTICE #3

- Don’t display status codes or error messages intended for developers to the user



https://blogs.adobe.com/creativecloud/xd-essentials-how-to-design-error-states-for-mobile-apps/

BEST PRACTICE #4

- Avoid blocking the UI whenever possible

BEST PRACTICES FOR

DOWNLOADING IMAGES

WORKING WITH IMAGES

- The iTunes Search API returns urls to images


```
{
    "resultCount": 20,
    "results": [
        {
            "wrapperType": "collection",
            "collectionType": "Album",
            "artistId": 262836961,
            "collectionId": 420075073,
            "amgArtistId": 861756,
            "artistName": "Adele",
            "collectionName": "21",
            "collectionCensoredName": "21",
            "artistViewUrl": "https://itunes.apple.com/us/artist/adele/262836961?uo=4",
            "collectionViewUrl": "https://itunes.apple.com/us/album/21/420075073?uo=4",
            "artworkUrl60": "https://is4-ssl.mzstatic.com/image/thumb/Music/v4/cf/7e/47/cf7e47a8
            "artworkUrl100": "https://is4-ssl.mzstatic.com/image/thumb/Music/v4/cf/7e/47/cf7e47a
            "collectionPrice": 10.99,
            "collectionExplicitness": "notExplicit",
            "trackCount": 12,
            "copyright": "℗ 2010 XL Recordings Ltd",
```


More data to download!

DOWNLOADING IMAGES

- Downloading an image is similar to downloading JSON
- Instead of converting `data` into a struct, you'll convert it into a `UIImage`

```
let task = URLSession.shared.dataTask(with: url) { data, _, error in
    guard let data = data, error == nil else {
        DispatchQueue.main.async { completion(nil, error) }
        return
    }

    if let image = UIImage(data: data) {
        DispatchQueue.main.async { completion(image, nil) }
    } else {
        DispatchQueue.main.async { completion(nil, error) }
    }
}
task.resume()
```

BEST PRACTICE #1

- Download images on an as-needed basis.
    - How do you know that a cell is about to be displayed on screen?

BEST PRACTICE #2

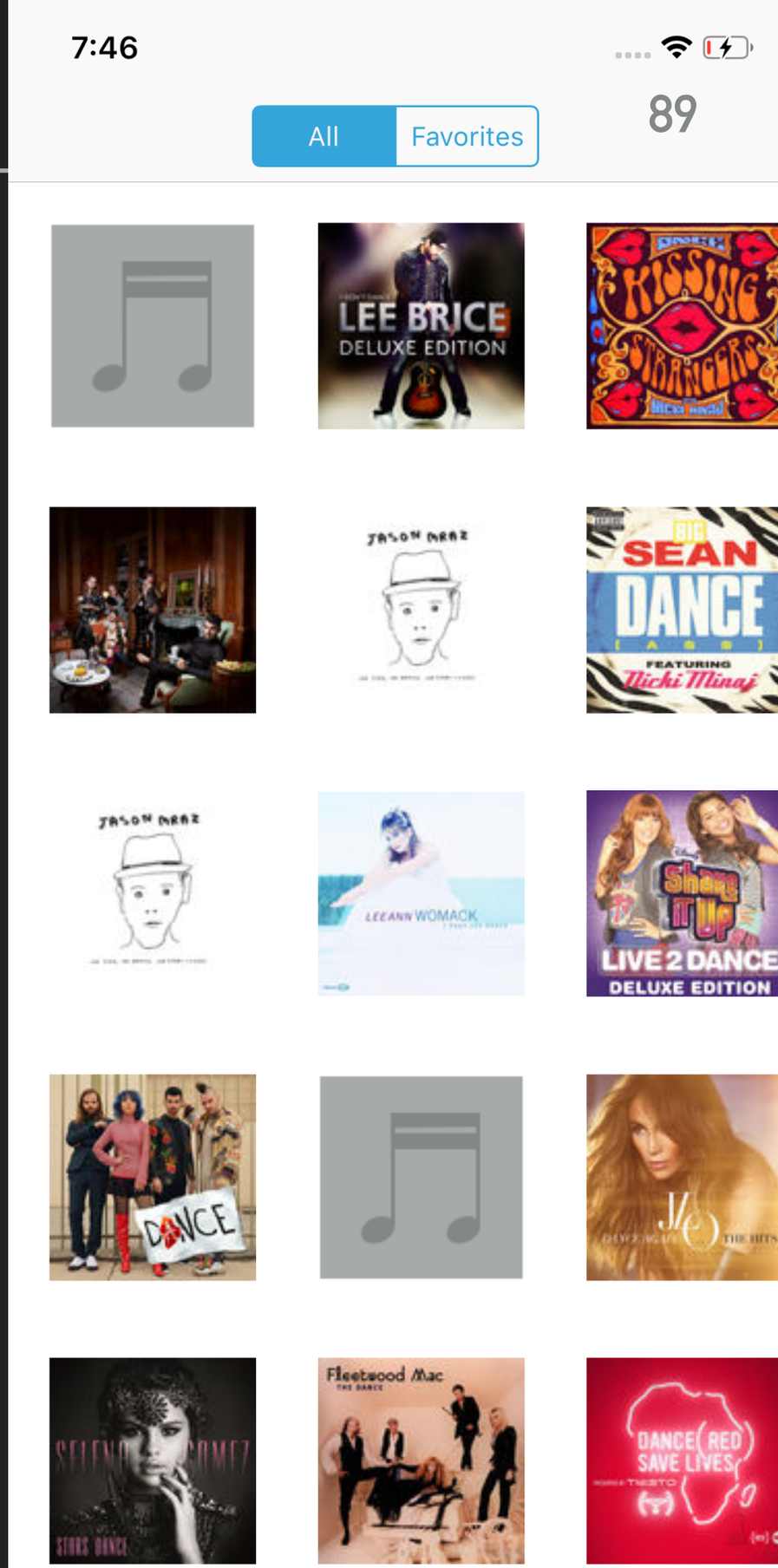
- Do not download images on the main queue.

```
let data = try! Data(contentsOf: url)
let image = UIImage(data: data)
```

THIS DOWNLOADS THE IMAGE ON THE MAIN QUEUE
(UNLESS YOU PUT IT ON A GLOBAL QUEUE)

BEST PRACTICE #3

- Show a placeholder image while the real image is downloading or if an error occurs.
- Consider overriding `prepareForReuse()`



INTRODUCING

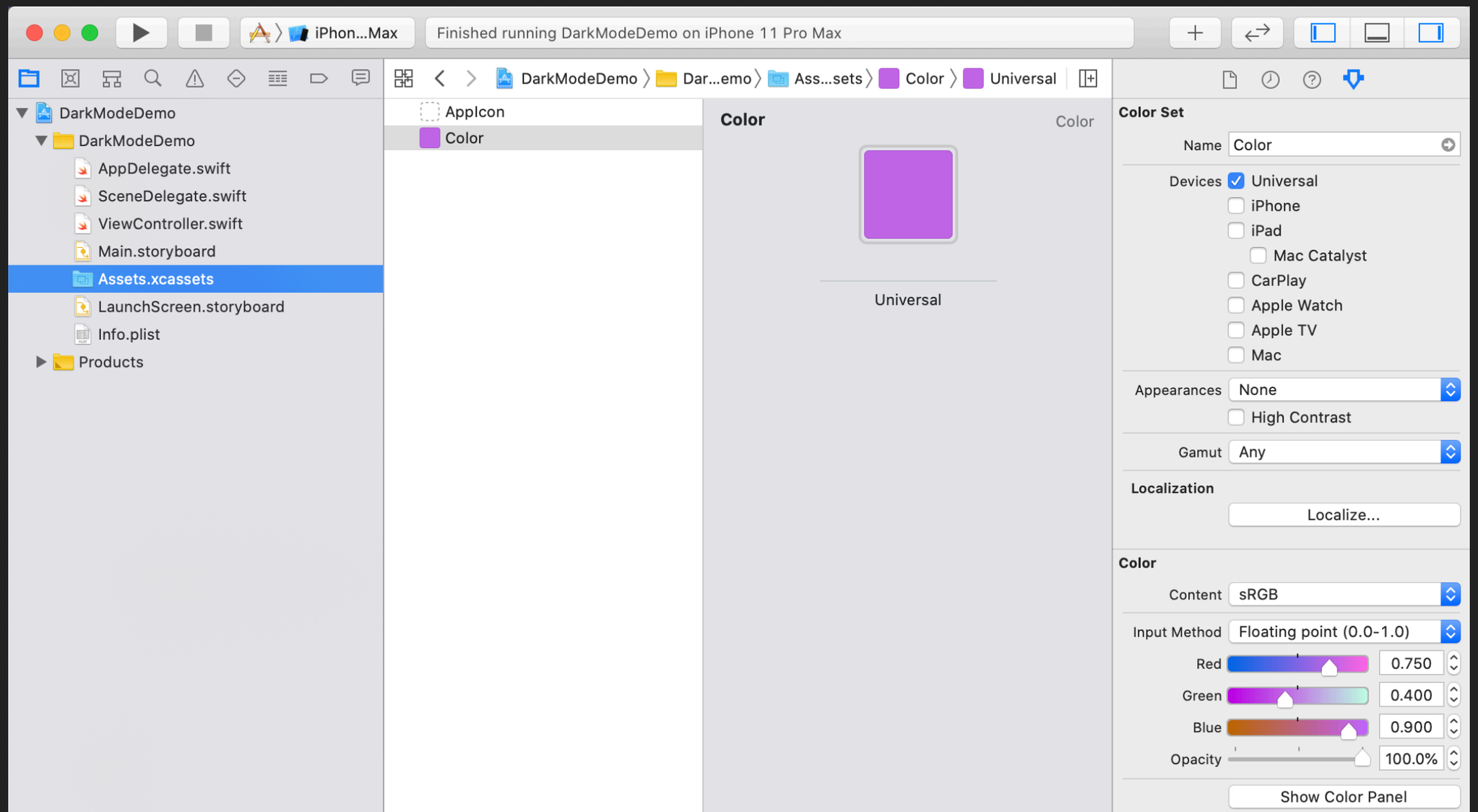
DARK MODE

COLORS (BEFORE IOS 13)

- Before iOS 13, UIKit included a number of standard colors
  - `UIColor.red`
  - `UIColor.blue`
  - `UIColor.green`
  - etc...

COLORS (BEFORE IOS 13)

- You can also create your own colors



COLORS (BEFORE IOS 13)

- You can also create your own colors

```
view.backgroundColor = UIColor(red: 52/255,
                               green: 100/255,
                               blue: 90/255,
                               alpha: 1.0)
```

DARK MODE

- In iOS 13, Apple introduced dark mode

DARK MODE

- New system colors introduced with iOS 13 make it easy for developers to support dark mode

https://nshipster.com/dark-mode/

Name	API	Light		Dark	
		Default	Accessible	Default	Accessible
Red	`systemRed`				
Orange	`systemOrange`				
Yellow	`systemYellow`				
Green	`systemGreen`				
Teal	`systemTeal`				

DARK MODE

- New semantic colors are similar to system colors
- They are named for their function rather than appearance

Label Colors

```
class var label: UIColor
```

The color for text labels that contain primary content.

```
class var secondaryLabel: UIColor
```

The color for text labels that contain secondary content.

```
class var tertiaryLabel: UIColor
```

The color for text labels that contain tertiary content.

```
class var quaternaryLabel: UIColor
```

The color for text labels that contain quaternary content.

Fill Colors

```
class var systemFill: UIColor
```

An overlay fill color for thin and small shapes.

```
class var secondarySystemFill: UIColor
```

An overlay fill color for medium-size shapes.

```
class var tertiarySystemFill: UIColor
```

An overlay fill color for large shapes

DARK MODE

- New semantic colors are similar to system colors
- They are named for their function rather than appearance

Label Colors

```
class var label: UIColor
```

The color for text labels that contain primary content.

```
class var secondaryLabel: UIColor
```

The color for text labels that contain secondary content.

```
class var tertiaryLabel: UIColor
```

The color for text labels that contain tertiary content.

```
class var quaternaryLabel: UIColor
```

The color for text labels that contain quaternary content.

Fill Colors

```
class var systemFill: UIColor
```

An overlay fill color for thin and small shapes.

```
class var secondarySystemFill: UIColor
```

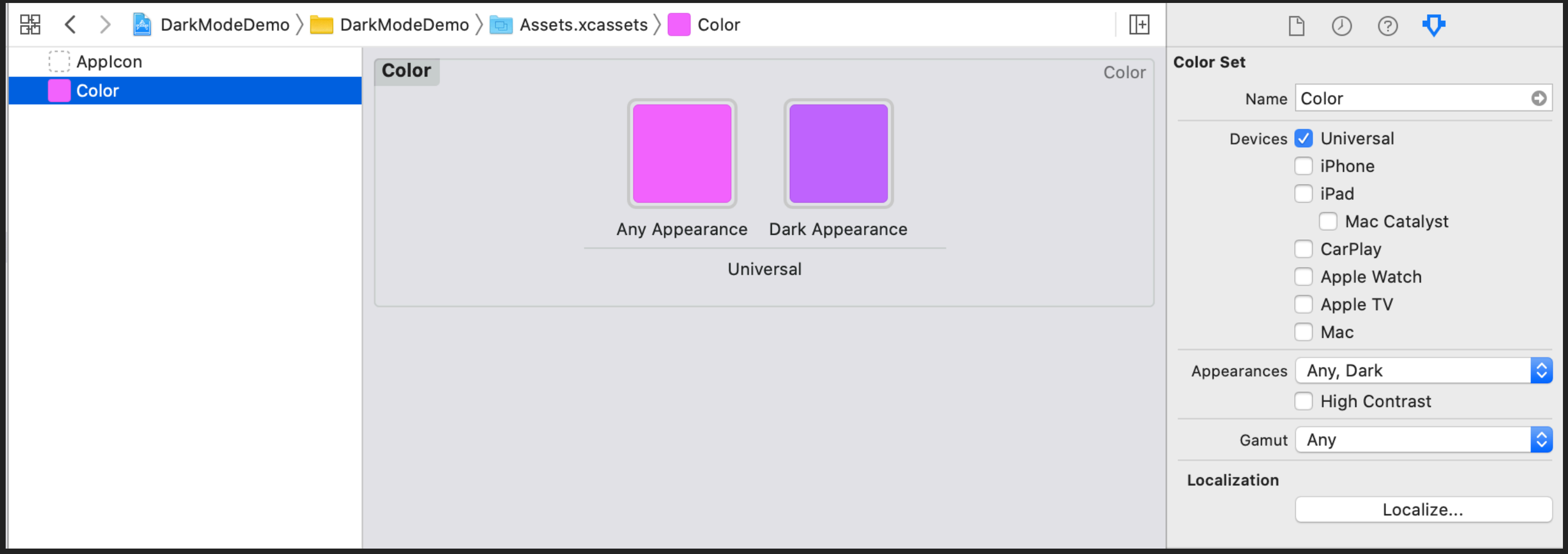
An overlay fill color for medium-size shapes.

```
class var tertiarySystemFill: UIColor
```

An overlay fill color for large shapes

DARK MODE

- If you need to create your own colors, you can add light and dark variations in the Assets Catalog



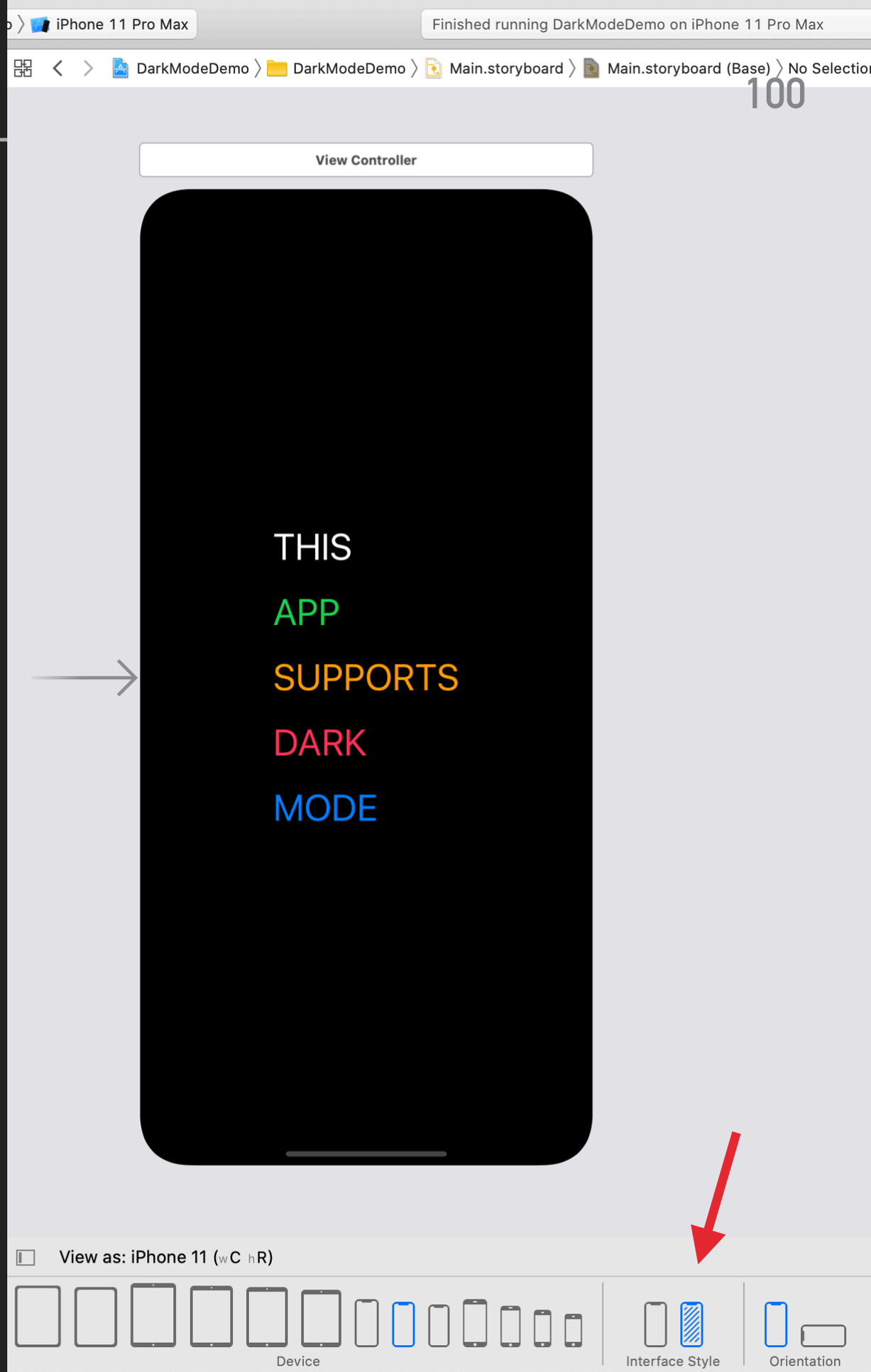
DARK MODE

- You can also create a dynamic color programmatically

```
let dynamicColor = UIColor { traitCollection in
    if traitCollection.userInterfaceStyle == .dark {
        return .white
    } else {
        return .black
    }
}
```

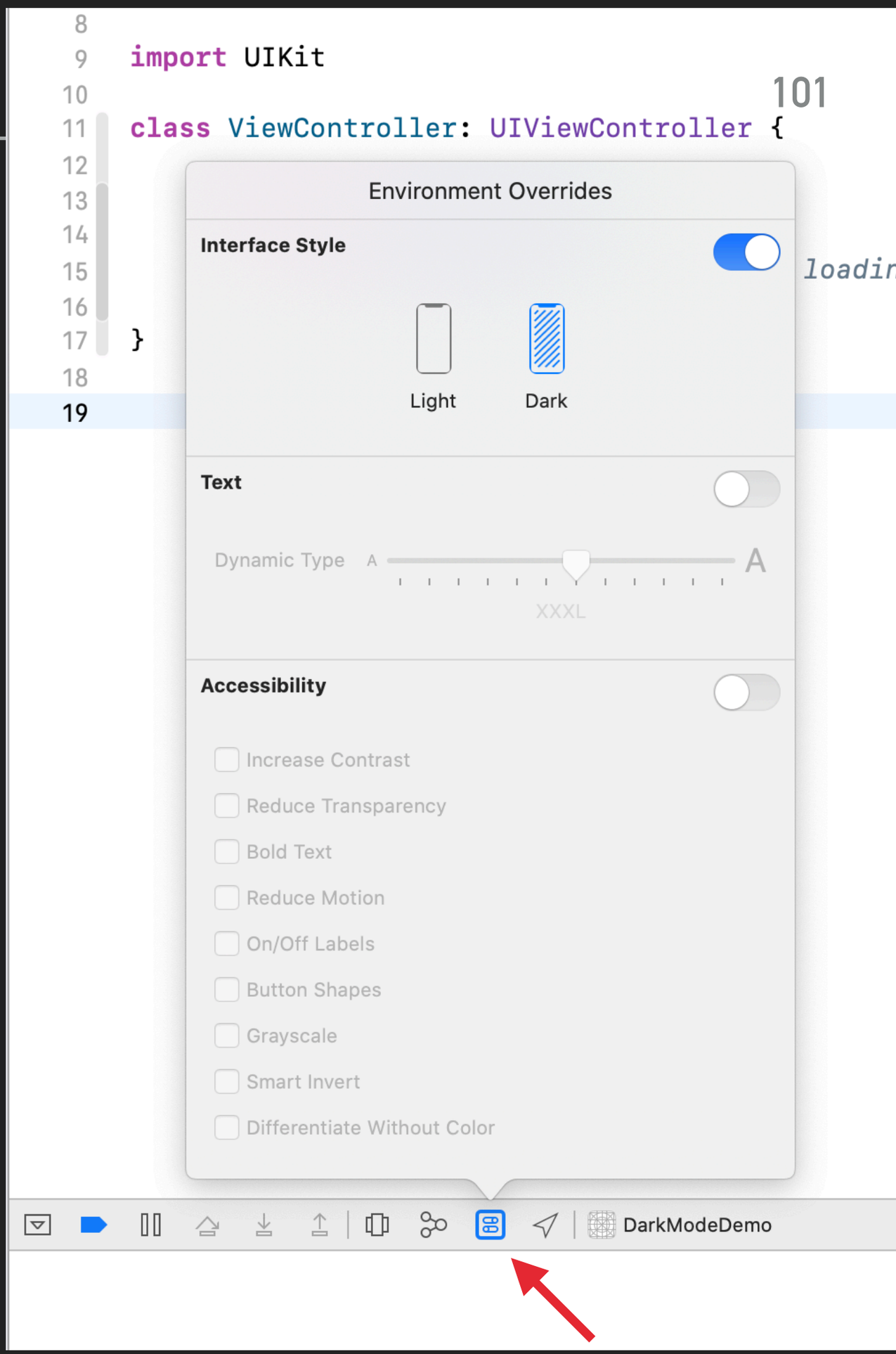
DARK MODE

- To preview dark mode in storyboard, use the Interface Style option



DARK MODE

- To preview dark mode in the simulator, open the Environment Overrides button



ASSIGNMENT #6

MOVIES

MOVIES

- Build an app that displays information about movies from the iTunes Search API.
- This assignment will use:
  - UICollectionViewController
  - UICollectionViewCompositionalLayout
  - UICollectionViewDiffableDataSource

MOVIES

- It will also give you the opportunity to use:
  - The delegate pattern
  - Best practices for downloading images
  - Dark mode

MOVIES

- Due Wednesday, February 19 at 5:29pm
- Post any questions to Slack